D3.2 Design and implementation of the acoustic processing unit

Author: Sebastian Uziel (IMMS), Thomas Elste (IMMS), Wolfram Kattanek (IMMS)

Contributor (s): Arturo Marquina (SOL), Victoria Portillo (COR), Marcella Scuccimarra (SEA)

Project Acronym: S4ECoB

Grant Agreement Number: 284628

Issue Date:	**M30**
Deliverable Number:	**D3.2**
Work Package Number:	**WP3**
Status:	**Final Version**

DISSEMINATION LEVEL	
X	PU = Public
	PP = Restricted to other programme participants (including the EC)
	RE = Restricted to a group specified by the consortium (including the EC)
	CO = Confidential, only for members of the consortium (including the EC)

Document History			
Version	**Date**	**Author**	**Description**
1	26.03.2012	IMM	ToC
2	28.06.2012	IMM	Preliminary Version
3	12.03.2014	IMM	Draft Version submitted for QCC
4	24.03.2014	Andrea Cavallaro as representative of D'Appolonia project team (DAP)	QCC Review
5	31.03.2014	IMM	Final Version

Disclaimer

The information proposed in this document is provided as a generically explanation on the proposed topic. No guarantee or warranty is given that the information fits for any particular purpose. The user thereof must assume the sole risk and liability of this report practical implementation. The document reflects only the author's views and the whole work is not liable for any empirical use of the information contained therein.

CONTENTS

TERMINOLOGY AND ABBREVIATIONS

BEMO	building energy management system optimizer = overall S4ECoB system
BMS	building management system
BEMO server	central component of each BEMO installation (usually 1 BEMO server per building / site); consists of several software components (e.g. data base, data fusioning module, GUI and BMS interface, gateway to occupancy sensor network, internet remote access) running on a dedicated PC / server or on PC / server which is already part of the BMS
APU	Acoustic Processing Unit
ASU	Audio Satellite Unit
FPGA	Field Programmable Gate Array
SoC	System on Chip
IC	Integrated Circuit
GPMC	General Purpose Memory Interface
ADAT	Alesis Digital Audio Tape
PTP	Precision Time Protocol

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

This deliverable is associated to T3.2 that intends to develop and implement the Acoustic Processing Unit (APU) as the main component of the occupancy sensor.

Each S4ECoB occupancy sensor consists of an Acoustic Processing Unit (APU), an energy-efficient but capable, adaptable and scalable embedded audio processing computer and up to 3 Audio Satellite Units (ASU) each connected with up to 8 microphones. The APU extracts and processes the audio streams transmitted by the ASUs in near real time and detects events to discriminate the level of occupancy in the corresponding room or area.

An overview over the APU architecture and the general information flow in the overall S4ECoB system solution are discussed within this deliverable.

The following components have been developed and are described in the present document:

- APU hardware platform, consisting of a mainboard, a FPGA-based extension board and the corresponding FPGA firmware. Housing details and hardware cost aspects are addressed as well.
- APU software platform, including an embedded operating system, a signal processing framework, a number of communication and synchronization components (for communication with ASU's and BEMO server) and several management functions like secure remote access and safe remote firmware update.

Furthermore, results of the technical system validation and corresponding tests, which had to be performed to validate the defined requirements from Deliverable D2.3, are reported in this deliverable. It can be summarized that the APU fulfills all of the technical requirements necessary for the realization of the S4ECoB demonstrator installations.

Finally, this deliverable contains the APU and occupancy sensor network installation and commissioning manual targeted at installation staff and IT personnel working at the demo site facilities.

1 INTRODUCTION

1.1 Purpose of this document

The S4ECoB system is aimed at monitoring and processing sounds and noises for an accurate determination of the types of occupancy and activities inside and outside smart buildings in order to improve the Building Energy Management (BEM) systems and in consequence to optimize the Energy efficiency in Buildings (EeB).

The S4ECoB system consists of two main components:

- Occupancy sensors with integrated Acoustic Processing Unit (APU) and up to 3 Audio Satellite Units (ASU) each connected with up to 8 microphones
- Building Energy Management Optimizer server (BEMO server)

The Acoustic Processing Unit (APU) is an energy-efficient but capable, adaptable and scalable embedded audio processing computer. The APU extracts and processes the audio streams in near real time and detects events to discriminate the level of occupancy in the corresponding room or area.

The development of the APU hardware and software is described in this document. Also the results of the technical validation and test procedures of the APU are described in this document.

1.2 Structure of the deliverable

This document is structured as follows:

- In Section 2 there is an overview of the S4ECoB system and the role of the APU in the overall system.
- In Section 3 the APU hardware components including the extension board and the FPGA firmware are explained in detail.
- In Section 4 the APU software platform and its components are described.
- In Section 5 the technical system validation report
- The APU and occupancy sensor network setup and management manual are described in Section 6.
- The conclusions are given in Section 7.

The deliverable is completed with a list of references and two appendices containing board schematics and the APU physical and electrical installation manual.

1.3 Relationship to the project objectives

The APU (Objective 1) is a basic component of the S4ECoB system. It extracts and processes the audio streams from the ASUs in near real-time, i.e. it detects and classifies acoustic events in order to discriminate between levels of occupancy in the monitored rooms and areas. The APU is a part of the occupancy sensor (Objective 2) that will be incorporated in the pilot buildings (Objective 7 from DoW).

1.4 Relationship to other deliverables and tasks

WP2, task 2.2: "BEM system and demo site buildings requirements", task 2.3: "Communication needs and acoustic processing unit requirements" and task 2.4: "Acoustic system and architecture requirements", have provided all the specifications and requirements in order to develop the APU hardware and software platform.

WP3, task 3.1: "Acoustic measurements and audio hardware", was responsible for the development of the ASU and had therefore a close relationship to the development of the APU in order to realize an optimized occupancy sensor. On the other side task 3.3: "Sounds processing and management system", task 3.4: "Monitoring of occupancy in the Audio Processing Unit" and task 3.5: "Production of the sound database for system retraining", need APU hardware and software design details in order to implement all acoustic processing functionality and system training capabilities. Therefore Deliverable D3.2 supports most of the other tasks of WP3.

WP4, task 4.1: "Integration of occupancy sensor network with BEMO server". The APU Gateway on the BEMO server is responsible for the communication with all the APUs inside the acoustic network. Implementation details of the communication protocol and the sound data management as described in Deliverables D4.1 and D3.2 have been inputs for both corresponding tasks.

Additionally, the installation and commissioning manual, as part of this deliverable, will be used during installation and operation of the S4ECoB system (WP6).

In Figure 1 the relation to previous and successive deliverables and tasks is shown.

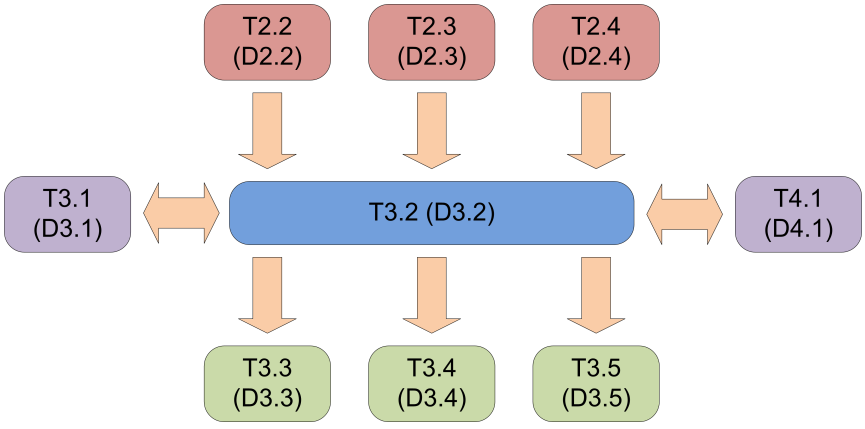

Figure 1: Relation of D3.2 to previous and successive tasks and deliverables

1.5 Contributions of Partners

IMMS had the main responsibility to prepare this document. SOL, SEA and COR were mainly contributing parts addressing the installation and commissioning process of the APUs.

2 APU SYSTEM OVERVIEW

The S4ECoB system is hierarchically designed and therefore consists of several components on different system and network levels (Figure 2). On the top level there's the BEMO server component that has only one instance per installation or building respectively. Due to its scalability the S4ECoB system supports an arbitrary number of APUs on the level of the acoustic or occupancy sensor network (the number of APUs is only limited by the number of available IPv4 network addresses in the corresponding subnet). Each APU is connected with up to three ASUs on the level of the audio sensing network.

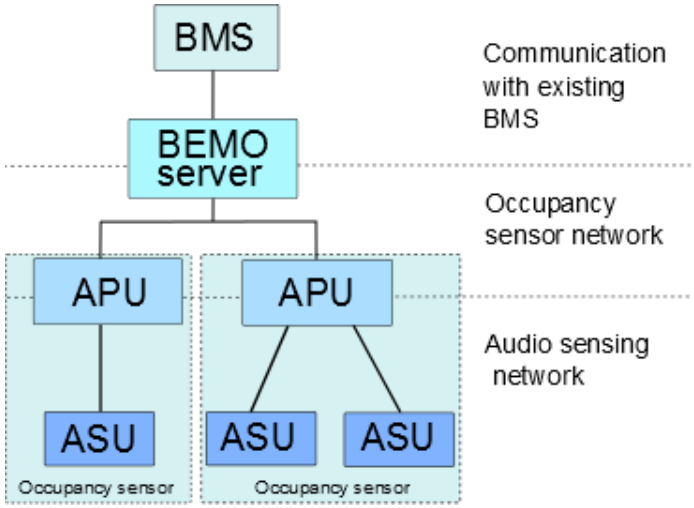

Figure 2: Hierarchical, network-oriented view of the overall S4ECoB system

In general the APU is an embedded signal processing and communication device. It is optimized for the processing of audio and acoustic data and for operation in a networked environment. Data processing and communication is performed continuously and the APUs therefore should operate 24/7 (while being permanently monitored by the APU Gateway component on the BEMO server). The overall S4ECoB flow of information is divided into blocks (distributed to several physical S4ECoB components) and corresponding data and control flow relations between particular blocks (Figure 3).

A more detailed description of the communication and sensing concept is given in the S4ECoB Deliverable D2.3.

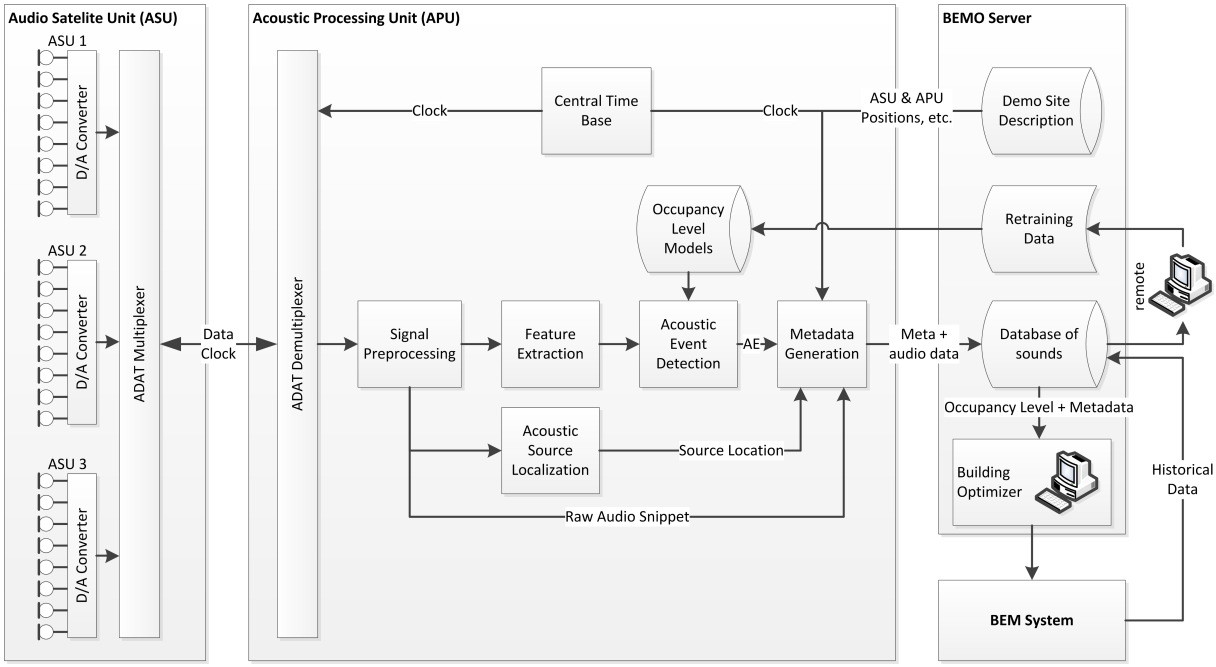

Figure 3: Data and control flow in the S4ECoB system

3 APU HARDWARE

3.1 Overview

The APU hardware platform consists of an embedded mainboard extended with an FPGA-based communication and signal processing extension board (Figure 4). The extension board receives the serial audio data stream from up to 3 ASUs. The ADAT encoded audio data (8 channels per APU) is first decoded and afterwards pre-processed in the FPGA. Typical pre-processing steps like e.g. down sampling and filtering of audio data lead to a reduction of the amount of data that has to be processed in later stages. Finally, the decoded and pre-processed data is transferred to the main board using a high-speed parallel communication interface (GPMC).

The main board is based on the commercially available embedded controller board Pandaboard. It was primarily chosen because of its CPU processing performance, its very good performance to power consumption ratio and the resulting low system costs in the case of a small number of APU prototypes. The main board serves as the hardware platform for the embedded operating system and the audio signal processing framework running on top of that (Section 4). Additionally, it is connected with the Ethernet-based occupancy sensor network.

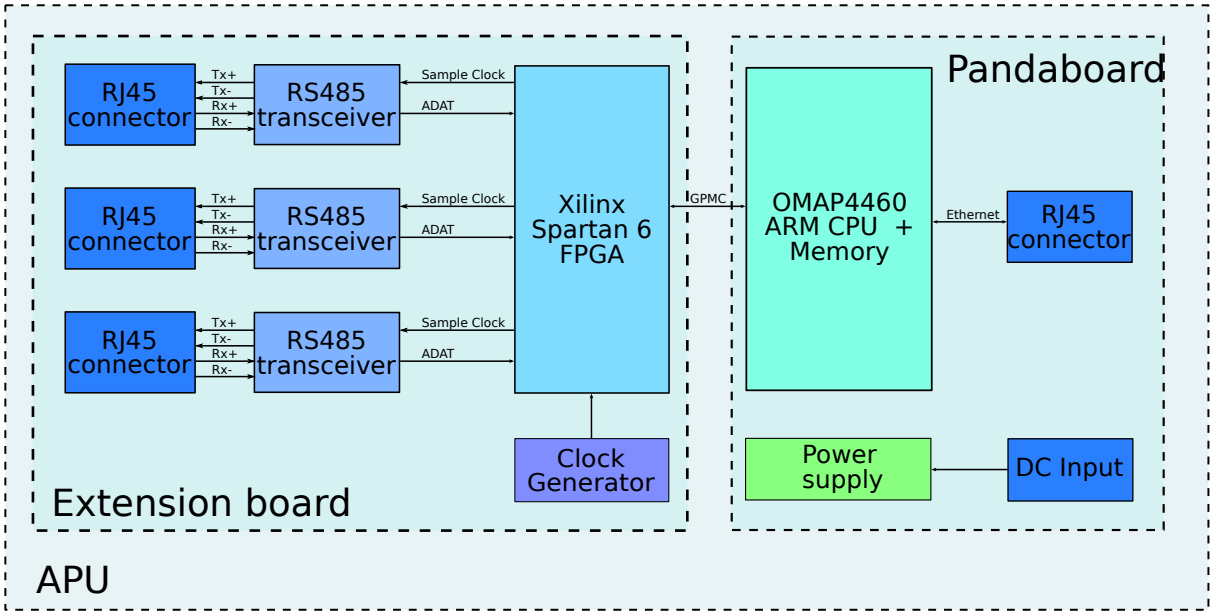

Figure 4: APU hardware architecture

The APU hardware architecture was already introduced in Deliverable D2.3. This document also contains a detailed discussion of the requirements leading to that particular architecture.

3.2 Mainboard

As described in Deliverable D2.3 the Pandaboard ES (Figure 5) was chosen as a base platform for the APU. This board is a low power, low cost single board computer based on the Texas Instruments OMAP4460 (Cortex-A9 multicore architecture) SoC.

Figure 5: Picture of the APU mainboard

A block scheme of the mainboard with the used interfaces and components is shown in Figure 6.

Power is delivered to the board by and external wall plug supply, with 5V voltage output and 2A maximum current.

The board provides an onboard 10/100Mb Ethernet interface which can be used for the occupancy sensor network interface. The expansion board is connected with the mainboard using the expansions connectors J3 and J6 (for detailed description see Section 3.3.3) of the mainboard.

The APU software components, Linux operating system device drivers and several userspace applications are stored on a SD card.

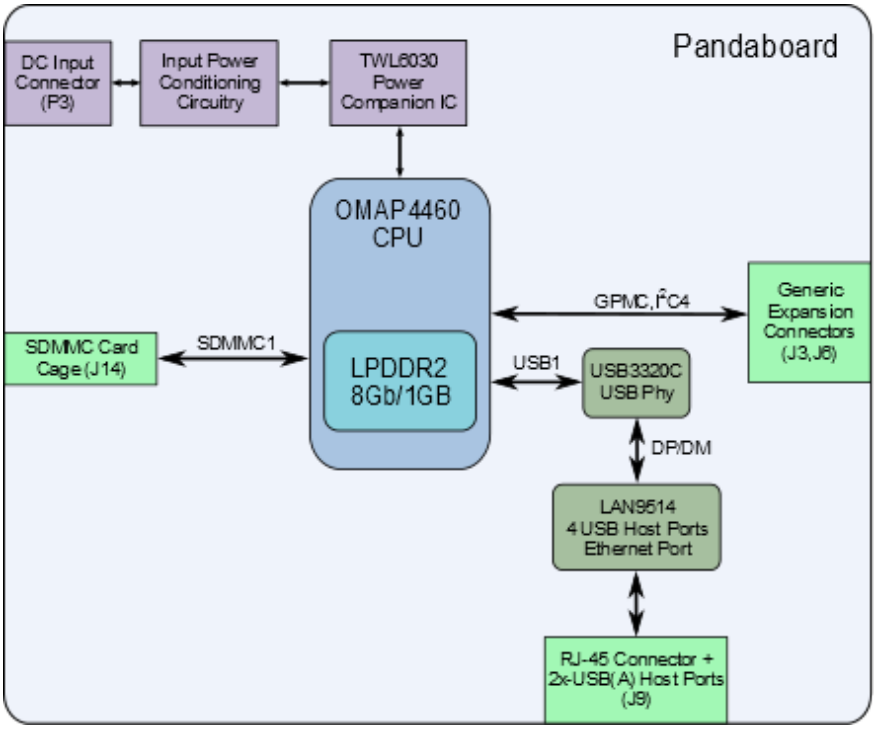

Figure 6: APU mainboard block scheme

The documentation, schematic and users guide of the Pandaboard are available at [1].

3.3 Expansion board

In Deliverable D2.3 the hardware and software requirements for the embedded acoustic processing unit and the APU hardware architecture is defined.

The main component of the expansion board is an FPGA, which is responsible for receiving, decoding and processing of maximum 24 audio channels from the ASUs and exchanging this audio data with the OMAP processor on the main board using the GPMC interface. In the following sections the components and interfaces of the expansion board are explained in detail.

3.3.1 Power supply

The APU expansion board is supplied via the main board expansion connectors. In Figure 7 a block diagram of the power circuitry is shown.

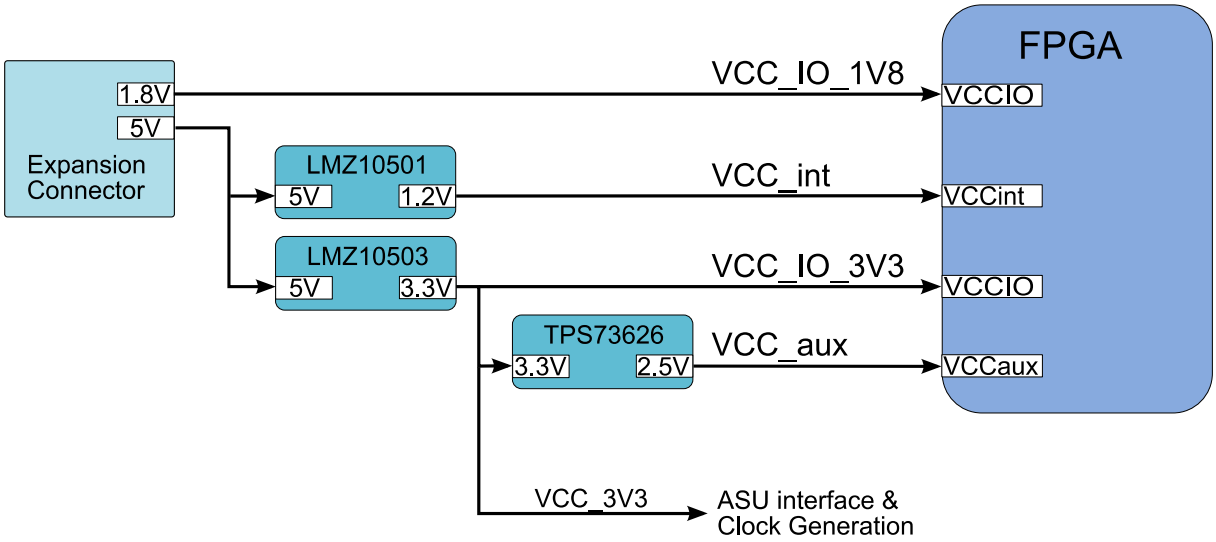

Figure 7: APU expansion board power circuitry block diagram

In Table 1 the voltage rails with the maximum current are shown.

Voltage rail	Voltage [V]	max. Current [A]	Description
VCC_IO_1V8	1.8	0,5	FPGA Bank 2 IO Voltage
VCC_int	1.2	1	FPGA core voltage
VCC_aux	2.5	0.4	FPGA auxiliary voltage
VCC_IO_3V3	3.3	0,5	FPGA Bank 0,1,3 IO Voltage
VCC_3V3	3.3	1	Supply for the other components of the expansion board

Table 1: Extension boards voltage rails

3.3.2 ASU interface

The signals from APU to ASU and vice versa are transferred using RS485 electrical standard, which specifies the electrical characteristics of the generator and the receiver. It does not recommend any protocol, only the physical layer.

Three ASU interfaces have been implemented, each consist of a RS485 Transceiver (SN65HVD37) with appropriate transient voltage suppressor (TVS) diodes and common mode inductors to protect the device from transient voltages resulting from electrostatic discharge (ESD), electrical fast transients (EFT) and lightning. The corresponding schematic diagram is included in the annex of this document.

For each ASU interface one LED is provided below the connector to display the connection status. A standard RJ45 connector is used, the pin assignment is listed in Table 2.

Pin	Signal	Description

1	TX+	positive driver output signal
2	TX-	negative driver output signal
3	RX+	positive receiver input signal
4	POE_VCC	POE supply voltage
5	POE_VCC	
6	RX-	negative driver input signal
7	POE_GND	POE signal ground
8	POE_GND	
S	shield	cable shield

Table 2: ASU interface connector pin assignment

3.3.3 Mainboard interface

The Pandaboard ES provides two 28-pin, 2,54mm through-hole expansion connectors, J3 and J6. The APU Expansion board is interfaced through these connectors with the processor mainboard. In Table 3 and Table 4 the pin assignments of the connectors are listed.

J3/X3 Pin	Pandaboard usage	Expansionboard usage	Description
1	Panda 1,8V	1,8V	1,8V supply
2	Panda 5V	5V	5V supply
3	GPMC_AD7	GPMC_AD7	GPMC Address/Data Bit 7
4	GPIO 140	DONE	Active-High signal indicating configuration is complete
5	GPMC_AD6	GPMC_AD6	GPMC Address/Data Bit 6
6	GPIO 156	GPMC_Int	GPMC Interrupt
7	GPMC_AD5	GPMC_AD5	GPMC Address/Data Bit 5
8	GPIO 155	GPIO 1	GPIO 1, Led V5 gn
9	GPMC_AD4	GPMC_AD4	GPMC Address/Data Bit 4
10	GPIO 138	GPIO 2	GPIO 2, Led V6 gn
11	GPMC_AD3	GPMC_AD3	GPMC Address/Data Bit 3
12	GPIO 136	GPIO 3	GPIO 3, Led V7 rt
13	GPMC_AD2	GPMC_AD2	GPMC Address/Data Bit 2
14	GPIO 139	GPIO 4	GPIO 4, Led V8 ge
15	GPMC_AD1	GPMC_AD1	GPMC Address/Data Bit 1
16	GPIO 137	GPIO 5	GPIO 5
17	GPMC_AD0	GPMC_AD0	GPMC Address/Data Bit 0
18	GPIO 135	GPIO 6	GPIO 6
19	GPMC_WE	GPMC_WE	GPMC Write Enable
20	GPIO 134	GPIO 7	GPIO 7
21	GPMC_OE	GPMC_OE	GPMC Output Enable

22	GPMC_AD15	GPMC_AD15	GPMC Address/Data Bit 15
23	I2C4_SDA	I2C_SDA	I2C data line
24	I2C4_SCL	I2C_SCL	I2C clock line
25	NC	-	
26	NC	-	
27	GND	GND	Digital Ground
28	GND	GND	Digital Ground

Table 3: Mainboard interface connector X3 pin assignment

J6/X6 Pin	Pandaboard usage	Expansionboard usage	Description
1		-	
2		-	
3		-	
4		-	
5		-	
6		-	
7		-	
8		-	
9	GPMC_AD14	GPMC_AD14	GPMC Address/Data Bit 14
10	GPMC_AD13	GPMC_AD13	GPMC Address/Data Bit 13
11		-	
12		-	
13		-	
14	GPIO 121	RDWR_B	Determines the direction of the D[x:0] data bus during Configuration
15		-	
16		-	
17	GPMC_AD12	GPMC_AD12	GPMC Address/Data Bit 12
18	GPMC_AD8	GPMC_AD8	GPMC Address/Data Bit 8
19	GPMC_WAIT	GPMC_WAIT	GPMC Wait
20	GPMC_AD9	GPMC_AD9	GPMC Address/Data Bit 9
21	GPIO 54	INIT_B	
22	GPMC_AD10	GPMC_AD10	GPMC Address/Data Bit 10
23	GPMC_CLK	GPMC_CLK	GPMC and Configuration clock
24	GPMC_AD11	GPMC_AD11	GPMC Address/Data Bit 11
25	GPMC_CS0	GPMC_CS0	GPMC Chip Select 0
26	GPMC_ALE	GPMC_ALE	GPMC Address Latch Enable
27	GPMC_CS1	GPMC_CS1	GPMC Chip Select 1
28	GPIO 59	PROG_B	Active-Low asynchronous full-chip FPGA reset

Table 4: Mainboard interface connector X6 pin assignment

The communication between FPGA and processor is realized using the General Purpose Memory Controller (GPMC) of the processor. This interface is also used to load the configuration bit stream into the FPGA during the startup of the APU.

3.3.4 Other components

An external clock generator circuit (PLL1707) provides a low jitter sample clock (50ps typical) and the FPGA system clock.

No-volatile data outside the SD card can be stored in an EEPROM on the expansion board, which is connected to the I2C bus 4 of the processor.

Three LEDs (V1, V2, V3) are located on the bottom side under the ASU interface connectors. Furthermore five LEDs (V4, V5, V6, V7, V8) are placed on the top side of the board, the LED V4 is connected with a series resistor to the 5V supply voltage. The other LEDs are connected to the FPGA and mapped to the GPIOs 155, 138, 136, 139 of the OMAP processor (see Table 3).

3.3.5 Dimensions and mounting

In Figure 8 the dimensions and the position of the mounting holes are shown. The position of the expansion connectors and the mounting holes fit to the main board positions.

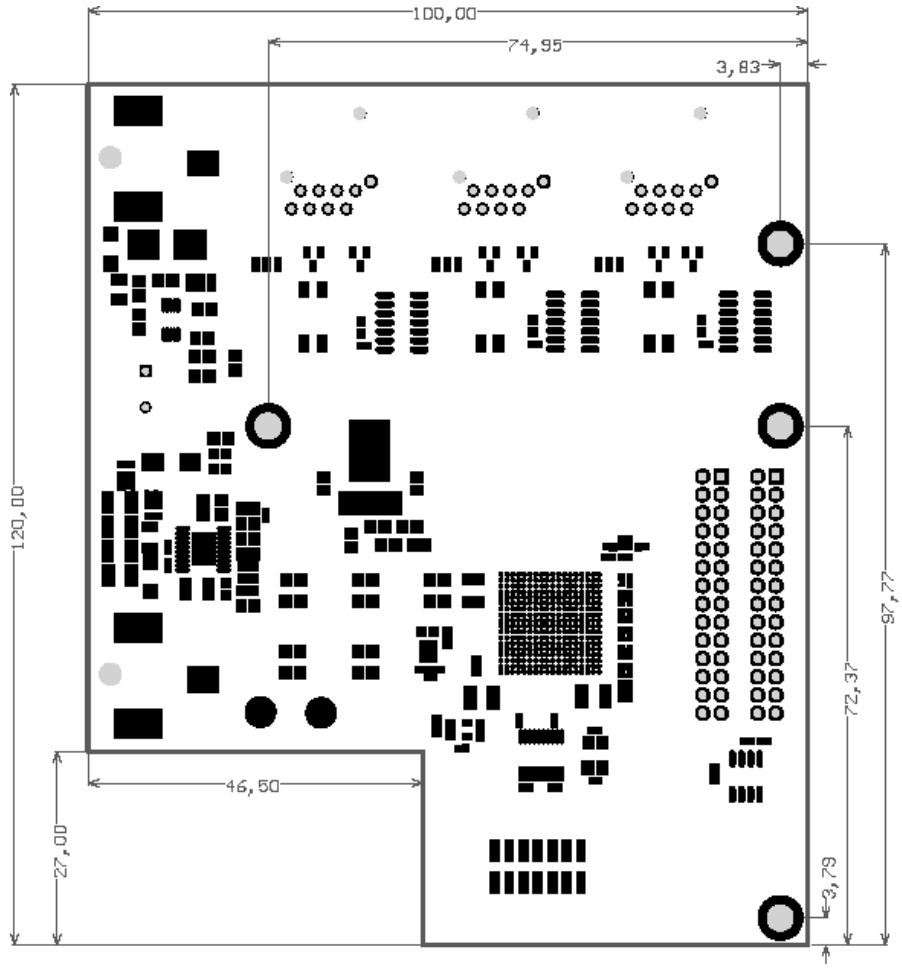

Figure 8: Expansion board dimensions

The expansion board can be mounted on top of the main board with 10mm spacers (Figure 9).

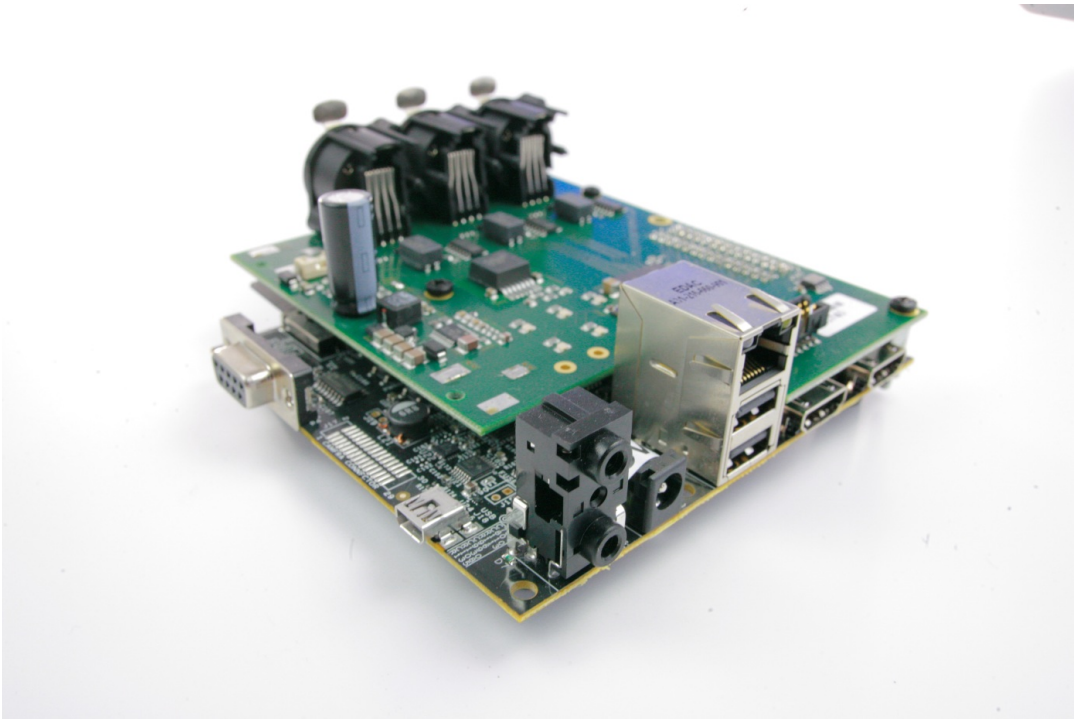

Figure 9: APU without housing

3.4 FPGA firmware

The FPGA firmware consists of three main components, namely the ASU interface, the pre-processing block and the processor interface. In Figure 10 the main components of the FPGA firmware are shown.

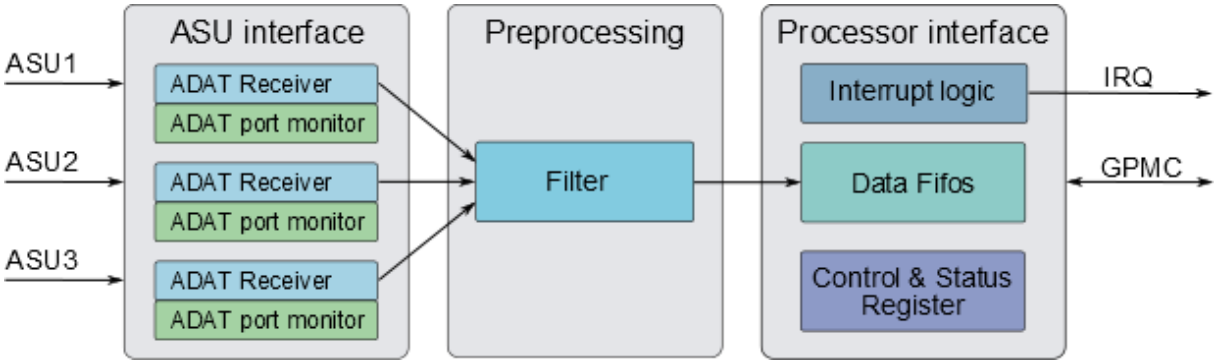

Figure 10: FPGA firmware block diagram

In the following sections the components are described in detail.

3.4.1 ASU interface

Up to three ASUs can be connected to one APU. Subcomponents of the ASU interface are three ADAT receivers and ADAT port monitors. One ADAT receiver decodes a serial audio data stream received

from an ASU and outputs parallel eight audio channels with 24bit data width. The ADAT port monitor detects if an ASU is connected to the corresponding APU interface or if an error in the communication occurs.

3.4.1.1 ADAT receiver

The ADAT receiver decodes the serial audio stream acquired by the ASU. The structure of the ADAT protocol is explained in Deliverable D2.3 (Section 3.1.2).

The ADAT receiver was developed using the hardware description language VHDL. The design was verified using Cadence Incisive simulator. A self-checking testbench with random stimulus generation was created to check several cases. Figure 11 shows a screenshot of the simulation output.

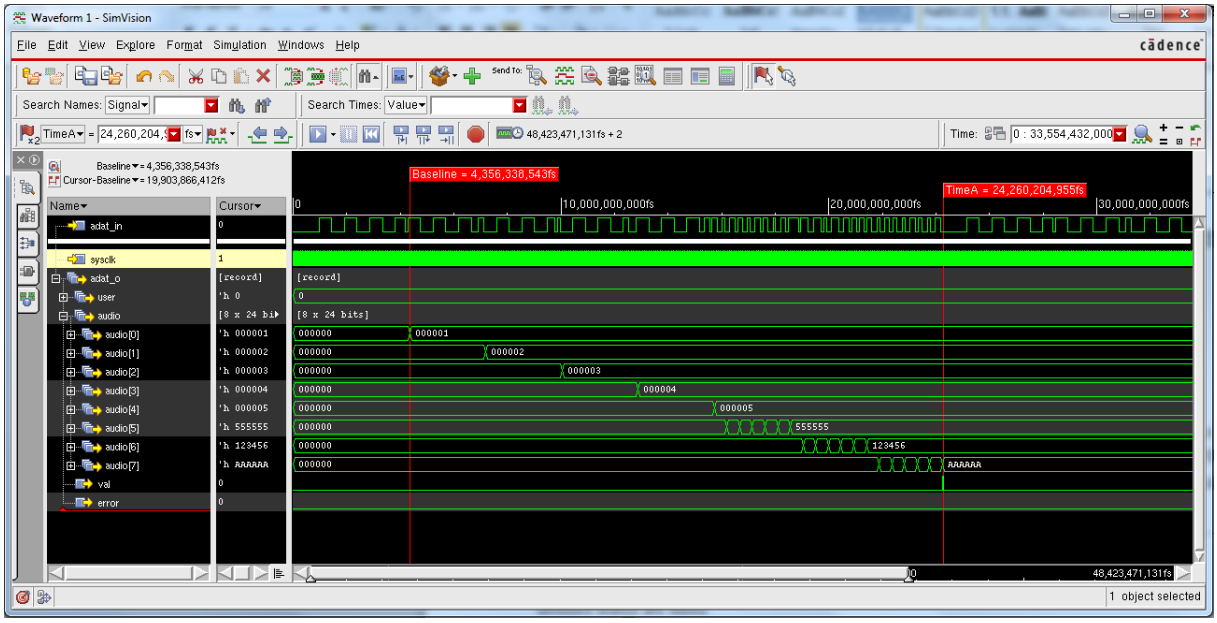

Figure 11: ADAT receiver simulation screenshot

3.4.1.2 ADAT Port Monitor

The port monitor detects the state of the corresponding ASU interface ADAT input. In Table 5 the different states are listed.

State	Value	Description
Init	0	Initial State, ASU Interface disabled
Enable	1	ASU interface enabled
Frame valid	2	Frame was decoded successfully, audio data valid

Edge error	3	No signal transition detected on the input, the ASU is disconnected
Frame error	4	Signal transition occurs on the input, but no valid frames could be decoded

Table 5: ASU interface ADAT connection states

The state values of the three port are accessible in the ADAT status register (see register description in Table 6).

Figure 12 shows the state diagram of the ADAT port monitor.

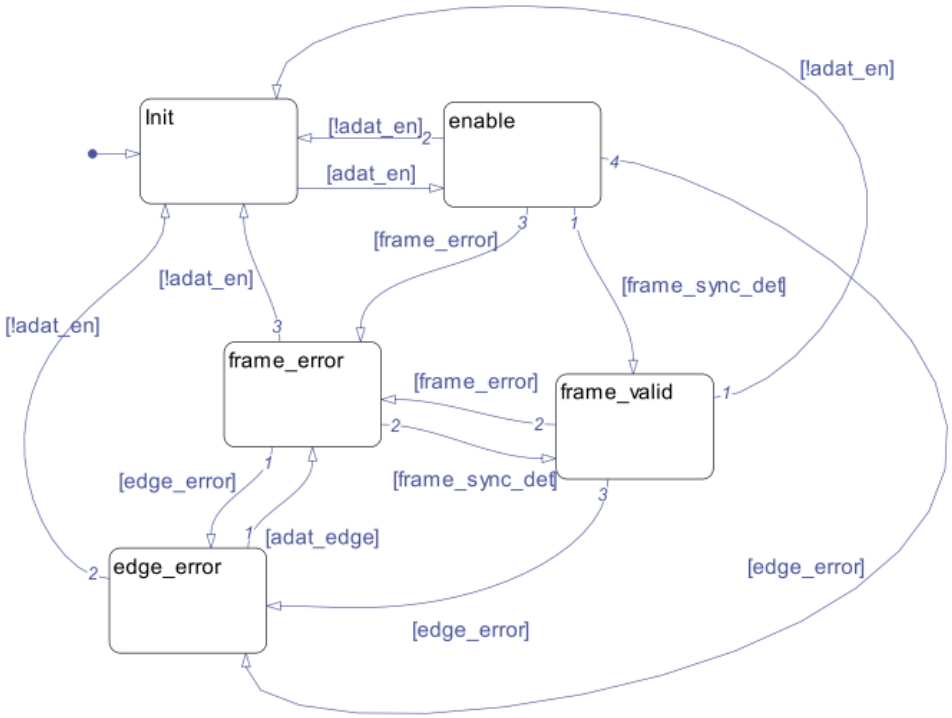

Figure 12: ADAT FSM

The port LED under the ADAT connector on the expansion board of each interface displays if a valid frame was decoded.

3.4.2 Pre-processing

The pre-processing algorithms were developed using Xilinx System Generator with Simulink® and MATLAB®. As described in Deliverable D2.4 (Section 3.2) a signal pre-filtering and microphone

calibration are algorithms that can be implemented in the FPGA. For these requirements a filtering in the frequency domain with configurable filter functions is implemented.

The filtering in the frequency domain is implemented using FFT convolution with the overlap-add method. The audio signal is converted from the time in the frequency domain using a Fast Fourier Transformation (512 point FFT, Radix 2) algorithm. Then, the data values are multiplied with the frequency response of the filter and transformed back to the time domain using an inverse FFT. The overlap-add method is used to enable the filtering with the FFT convolution of a very long audio signal. In this method the input signal is split into non overlapping segments and padded with zeros, and after that filtered as described above. The output signal is calculated by adding the overlapping segments [2].

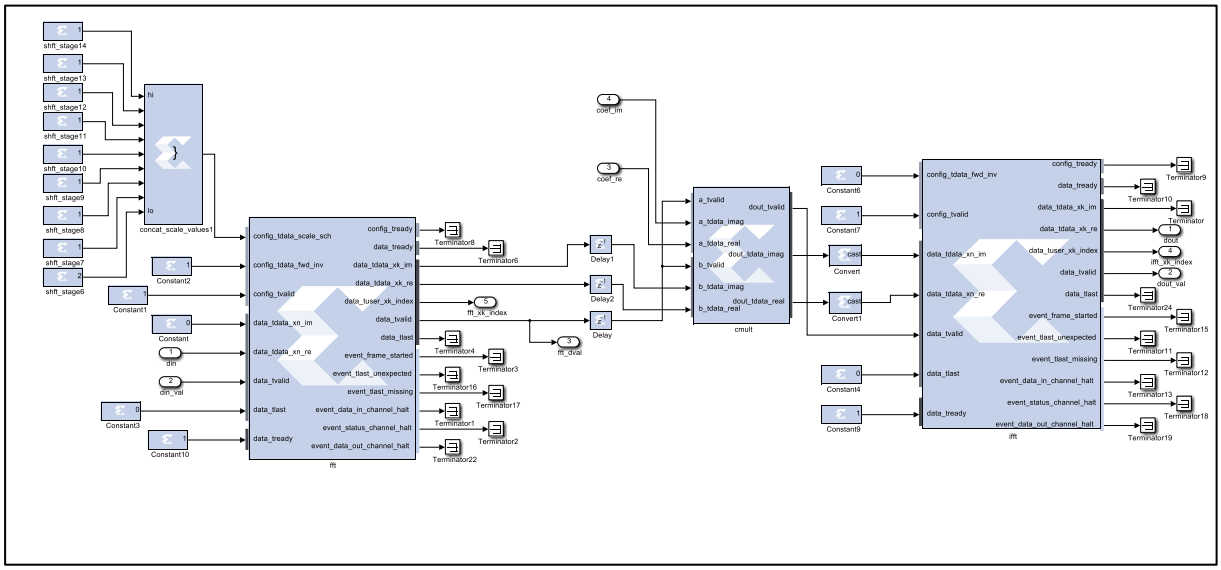

Figure 13: FTT filter Simulink model

In Figure 13 the Simulink model of the filter block is shown. All 24 audio channels were buffered, sequentially processed and afterwards transferred to the processor interface.

3.4.3 Processor interface

The FPGA is connected to the General Purpose Memory Controller (GPMC) of the OMAP processor. The 16 bit synchronous address / data – multiplexed mode of the GPMC is used. It features synchronous read and write as well as burst read access to the FPGA. The audio data has a resolution of 24 bit. The GPMC interface supports data transfers which are a multiple of 16 bit, therefore one audio sample has to be split in two 16 bit values.

For each of the 24 audio channels one 16 bit width and 1023 word depth FIFO is implemented to store audio data and enable a buffered block transfer to the processor. Several control and status registers (see Table 6 for detailed description) have been implemented.

To synchronize FPGA data capture and the transfer to the processor an interrupt line is used. An interrupt is generated if the status of one ASU interfaces changes or if the audio data FIFOs are half full

and a transfer to the processor can be started. The interrupt sources can be masked with the interrupt mask register and the source of the interrupt can be read out from the interrupt status register (see Table 6 for details).

In Figure 14 the structure of the processor interface is shown.

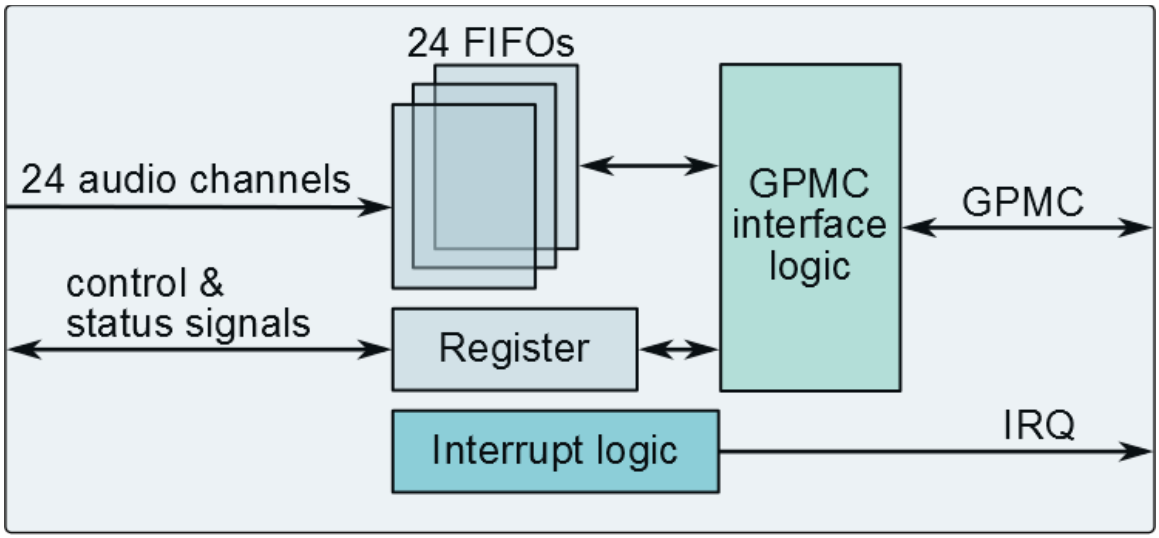

Figure 14: Processor interface block diagram

In Table 6 the implemented registers are described.

Register	Address	Bit	Type	Description
int_mask	0x0000		rw	Interrupt mask register
fifo_int_en		0		Fifo interrupt enable
adat_stat_en		1		ADAT status interrupt enable
rec_en	0x0001		rw	Record enable register
		0		ASU interface 1 receiver enable
		1		ASU interface 2 receiver enable
		2		ASU interface 3 receiver enable
clk_en	0x0002		rw	Clock enable register
		0		ASU Interface 1 sample clock enable
		1		ASU interface 2 sample clock enable
		2		ASU interface 3 sample clock enable
adat_stat	0x0003		r	ADAT status register
		3:0		ASU interface 1 ADAT status value
		7:4		ASU interface 2 ADAT status value
		11:8		ASU interface 3 ADAT status value

int_stat	0x004		r	Interrupt status register
		0		Fifo interrupt
		1		ASU interface 1 ADAT status interrupt
		2		ASU interface 2 ADAT status interrupt
		3		ASU interface 3 ADAT status interrupt

dev_dna1	0x0009		r	
dev_dna2	0x000A		r	
dev_dna3	0x000B		r	
dev_dna4	0x000C		r	
ver_dm	0x000D		r	
ver_y	0x000E		r	
ver_t	0x000F		r	

audio_ch1	0x0010 - 0x0200		r	Audio channel 1 data register
audio_ch2	0x0210 - 0x0400		r	Audio channel 2 data register
audio_ch3	0x0410 - 0x0600		r	Audio channel 3 data register
audio_ch4	0x0610 - 0x0800		r	Audio channel 4 data register
audio_ch5	0x0810 - 0x0A00		r	Audio channel 5 data register
audio_ch6	0x0A10 - 0x0C00		r	Audio channel 6 data register
audio_ch7	0x0C10 - 0x0E00		r	Audio channel 7 data register
audio_ch8	0x0E10 - 0x1000		r	Audio channel 8 data register
audio_ch9	0x1010 - 0x1200		r	Audio channel 9 data register
audio_ch10	0x1210 - 0x1400		r	Audio channel 10 data register
audio_ch11	0x1410 - 0x1600		r	Audio channel 11 data register
audio_ch12	0x1610 - 0x1800		r	Audio channel 12 data register
audio_ch13	0x1810 - 0x1A00		r	Audio channel 13 data register
audio_ch14	0x1A10 - 0x1C00		r	Audio channel 14 data register
audio_ch15	0x1C10 - 0x1E00		r	Audio channel 15 data register
audio_ch16	0x1E10 - 0x2000		r	Audio channel 16 data register
audio_ch17	0x2010 - 0x2200		r	Audio channel 17 data register
audio_ch18	0x2210 - 0x2400		r	Audio channel 18 data register
audio_ch19	0x2410 - 0x2600		r	Audio channel 19 data register
audio_ch20	0x2610 - 0x2800		r	Audio channel 20 data register
audio_ch21	0x2810 - 0x2A00		r	Audio channel 21 data register
audio_ch22	0x2A10 - 0x2C00		r	Audio channel 22 data register
audio_ch23	0x2C10 - 0x2E00		r	Audio channel 23 data register
audio_ch24	0x2E10 - 0x3000		r	Audio channel 24 data register

Table 6: Register description

3.4.4 Synthesis results

The design was synthesized, placed and routed using Xilinx ISE software. In Table 7 the used FPGA resources are listed:

Ressource	Used	Available	Utilization
Slices	3048	6822	44%
DSP 48A	36	58	62%
Block RAM	114	116	98%

Table 7: FPGA resource utilization

As shown in Table 7 nearly 100% of the available FPGA block RAM is used. This is due to the fact that the audio data has to be buffered for the pre-processing and the transfer to processor.

3.5 Housing

A Rittal polycarbonate housing (PK9517.100) with IP66 protection class and a transparent cover plate was selected for the APU. In Figure 15 a drawing and in Table 8 the dimension of the APU housing is shown.

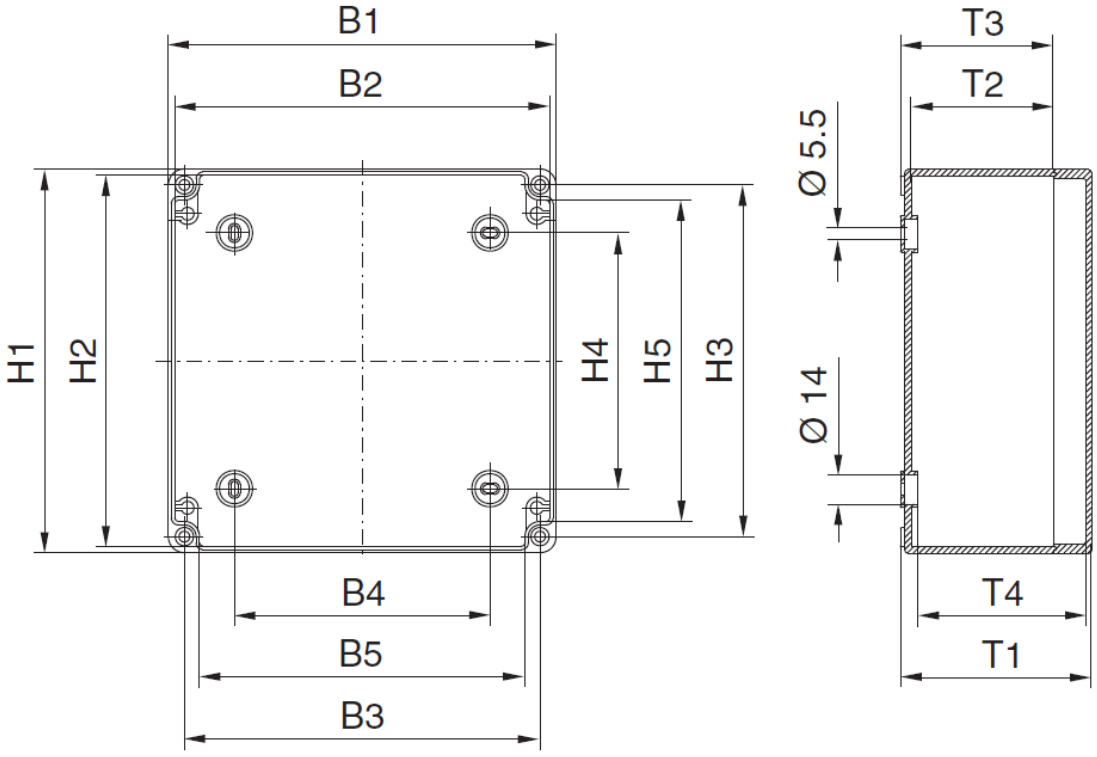

Figure 15: APU housing drawing

Dimension	Value mm	Description
B1	182	Housing width
B2	175	Internal housing width
B3	167	Wall fastening distance outside sealing
B4	120	Wall fastening distance inside the housing
B5	152	Usable width
H1	182	Housing height
H2	173	Internal housing height
H3	165	Wall fastening distance outside sealing
H4	120	Wall fastening distance inside the housing
H5	128	Usable height
T1	90	Complete housing depth
T2	63	Usable depth without cover plate
T3	71	Housing depth without cover plate
T4	75	Usable depth

Table 8: APU housing dimensions

The ASU interface connectors are mounted on the expansion board and can be screwed on the APU housing. A RJ45 feed through receptacle, Neutrik NE8FDP is mounted on the housing and connected with a 0.25m cat5e patch with the Ethernet port of the main board. A panel mount DC input jack 2.5 x 5.5 mm is connected to the DC input of the main board. The APU with housing has a weight of 708g.

The external interfaces are described in the commissioning manual (Annex B).

3.6 Costs

The costs per APU prototype are listed in Table 9.

Part	Price in a lot of 10
Mainboard	150 €
Expansion board	250 €
Initial production costs (Σ 820 €)	82 €
housing, power supply	40 €
total	**522 €**

Table 9: Costs for the APU prototype

The overall costs for the APU prototype could be massively decreased due to the fact of the well-known scaling effect when producing larger quantities of electronic devices. Here, we refer to information from a manufacturer of electronic boards and components as a reference. Estimations were given in the context of another R&D project [3] dealing with a device comparable with the APU main board (we assume that this alternative main board could be potentially used in the future due to APU SW optimizations and resulting reduced performance requirements). Another option would be a new

designed custom board using e.g. a recently presented Xilinx Zynq SoC (containing both a microprocessor and a FPGA) [4]. An estimated calculation based on these options is listed in Table 10.

	Cost-optimized APU mainboard		Zynq-based custom board	
Quantity	10	1000	10	1000
Mainboard	100 €	25 €	500 €	120 €
Expansion board	250 €	60 €		
Housing	40 €	20 €	40 €	20 €
total	**390 €**	**105 €**	**540 €**	**140 €**

Table 10: Estimated costs per APU at larger quantities

4 APU SOFTWARE

4.1 Software architecture

The software stack of the APU is composed of two boot loaders, an embedded Linux operating system, device drivers and several user-space applications. The complete software stack is stored on a SD flash card. The partitioning scheme of this flash card is listed in Table 11.

Partition	Content
1	First and second stage boot loader, boot loader configuration.
2	Root file system for rescue mode.
3	Root file system.

Table 11: APU SD card partitioning scheme

The basic root file system (based on the Ångström distribution for embedded devices) is created using the OpenEmbedded build framework [5]. By means of a customized configuration several additional libraries and programs necessary for APU operation are added automatically during the build process of the root file system by OpenEmbedded. This includes for example the Qt framework, the protobuf library for network communication, the Julius continuous speech recognition decoder software, the precision time protocol daemon, the watchdog daemon and several other utilities.

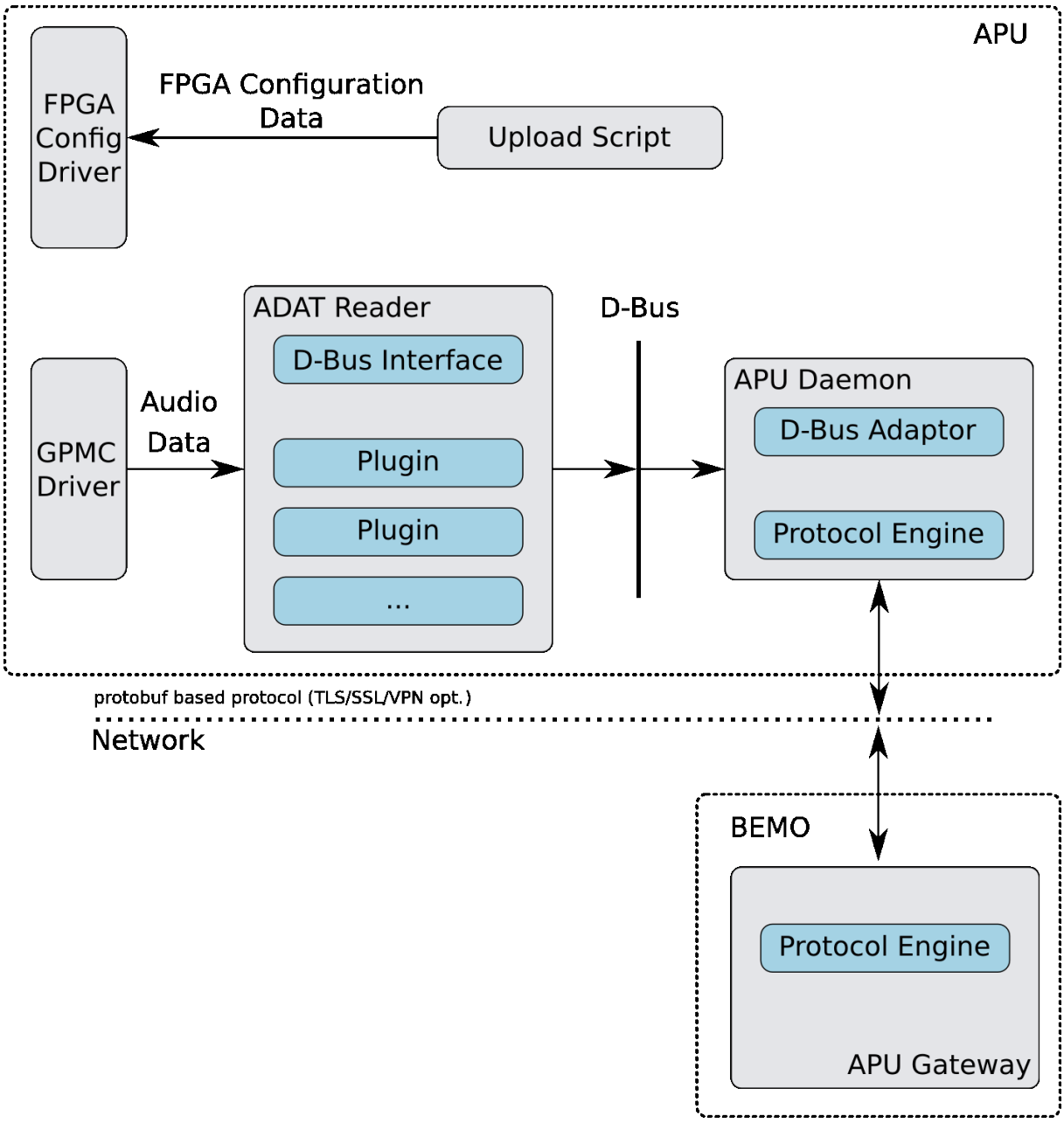

Figure 16: APU software overview

To this root file system several APU specific software components are added. Figure 16 shows an overview about these components responsible for connecting to the occupancy detection network, for audio data acquisition, processing and transmission of sensor data results.

The first part of the APU software stack has to take care of the FPGA configuration process. An upload script transfers the bitstream configuration data using a specialized device driver into the FPGA.

The main part of the software stack is responsible for audio data processing. Incoming ADAT audio streams are captured and decoded by the FPGA and transferred to the OMAP CPU using its General

Purpose Memory Controller (GPMC) in conjunction with a Linux device driver. Reception and processing of the data is done by the AdatReader application. Results (for example occupancy levels) of the audio processing algorithms are transferred to the APU Daemon, which is responsible for the integration of the APU into the occupancy sensor network and communicates with the APU Gateway running on the BEMO server.

The APU specific part of the software stack amounts to approximately 4800 lines of code, with the distribution among programming languages as shown in Table 12 (generated by the tool SLOCCount).

Totals grouped by language (dominant language first):	
cpp	3018 (62.16%)
ansi	1782 (36.70%)
sh	55 (1.13%)

Table 12: APU software used programming languages

4.2 FPGA communication

4.2.1 FPGA configuration driver

This device driver is responsible for enabling the configuration upload to the FPGA. To allow dynamic configuration changes and also to ensure the update capability of APUs installed at the demo sites this configuration method was chosen in favor of configuring the FPGA statically by EEPROM or Flash memory.

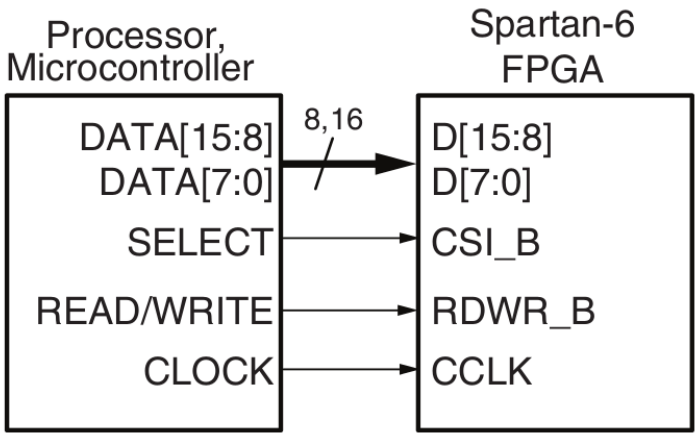

Figure 17: FPGA configuration connection scheme

Using this driver, bitfile data is transmitted over the GPMC interface of the OMAP4 CPU in combination with the x16 SelectMAP parallel upload mode of the Spartan6 FPGA. Figure 17 (from [6]) shows the pin configuration used in this mode. For data 0-15, chip select and clock signals the corresponding GPMC lines of the CPU are used. The RDWR signal is generated using an extra GPIO.

The parallel SelectMAP interface allows a very fast configuration process. As the FPGA configuration is volatile and lost after an APU power cycle, this helps to speed up the APU's boot process as the configuration process has to be carried out before audio data reception is possible.

This driver was implemented using code from the Armadeus Project [7], which already provides bitfile handling and upload logic so only adding the GPMC back end for OMAP CPUs was necessary.

4.2.2 FPGA upload script

FPGA bitfile upload is carried out using the script /opt/s4ecob/bin/fpgaconf.sh (on the APU root file system) which uses the character device interface provided by the configuration driver (see previous section). The script can be executed manually with the bitfile which should be uploaded as first parameter. Or the corresponding systemd service (Section 6.3.3) can be used which will always upload the file /opt/s4ecob/res/s4eeb_top.bin.

4.2.3 GPMC interface driver

Low level data transfer between FPGA and CPU is handled on the CPU side using a Linux driver module. The driver module configures the GPMC interface of the OMAP CPU for synchronous burst mode transfer. This allows reading up to 16 double words of data in one transfer cycle thus greatly increasing the data rate. Using this setting the driver can access the register interface implemented in the FPGA via IO memory operations. To improve performance and offload CPU the audio data transfer is accelerated utilizing the CPU's DMA controller for (IO) memory to memory transactions. Only configuration and status registers of the FPGA are read or written with DMA controller support.

To synchronize FPGA data capture and the drivers data transfer an interrupt line is used.

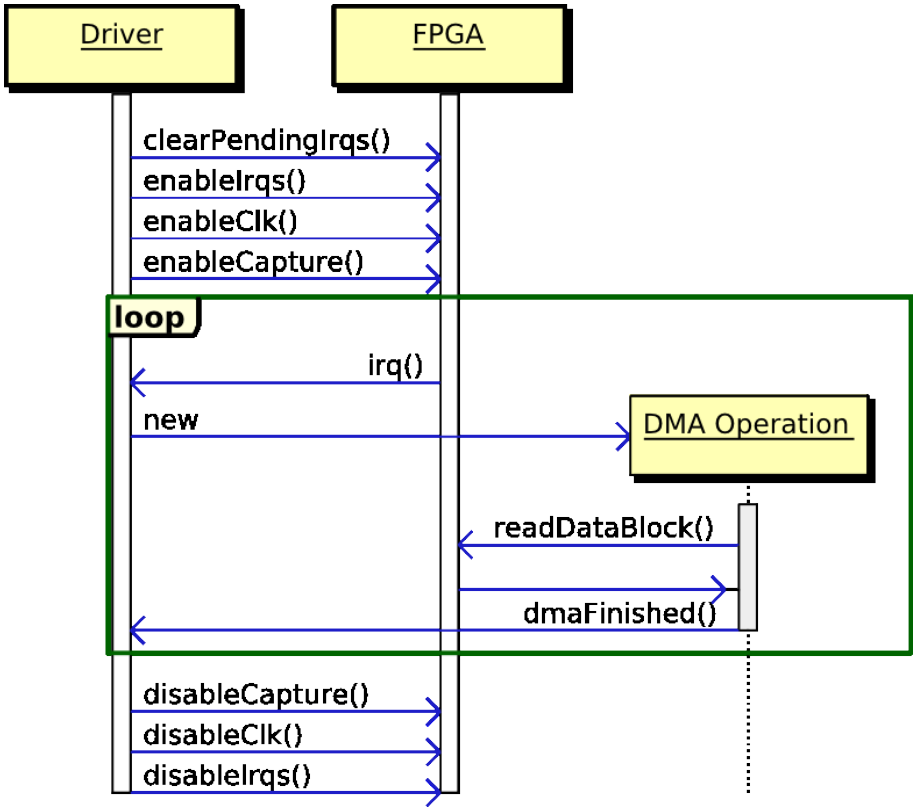

Figure 18: Processor FPGA communication sequence diagram

Figure 18 shows a sequence diagram of audio data capture operation between driver and FPGA. Starting with register accesses from the driver to the FPGA to setup interrupt, clock and record flags, the driver triggers the start of capture operation. After that, if the FPGA has gathered enough data in its internal FIFO, it sends out an interrupt request to the CPU. Following this interrupt the driver starts a new DMA operation which transfers one block of audio data from the FPGAs FIFO into the audio buffer memory of the driver. This is repeated for every block of audio data. If the data capture process is to be canceled the driver disables capture, clock and interrupt flags in the corresponding FPGA registers.

The userspace interface of the driver is realized using a Linux character device. After opening the device file /dev/fpga_gpmc a userspace program may use ioctl, read, write and mmap system calls to communicate with the driver.

Four different ioctl-calls are supported, listed in Table 13. With these single FPGA register access and controlling the DMA operation is possible.

IOCTL	Function
FPGA_GPMC_REG_READ	Reads a single FPGA-Register
FPGA_GPMC_REG_WRITE	Writes a single FPGA-Register
FPGA_GPMC_START_DMA	Starts DMA data transfer from FPGA-Fifo
FPGA_GPMC_STOP_DMA	Stops DMA data transfer

Table 13: FPGA write and read access functions

Read and write calls provide a way of synchronizing DMA data transfer and microphone array events detected by the FPGA to the calling user space application. Three different operations are possible:

- Blocking read with the size of “struct fgpa_gpmc_bufstat”. This is synchronized to DMA events and will unblock if a DMA transfer by the driver was completed and new audio data is available with status information of the audio buffer.
- Blocking read with the size of “size of (uint16_t)”. This is synchronized to events detected by the FPGA (like adding or removing an ASU) and will unblock if such an event was detected returning the new event status.
- Write calls with the size of “struct fgpa_gpmc_bufstat”. These will tell the driver when the userspace program has finished processing an audio buffer.

The last operation supported by the driver is the mmap system call. This maps the drivers audio buffer memory into the calling programs process memory allowing direct read access. This avoids unnecessary copy operations of audio data saving memory bandwidth of the CPU. As drawback to this method the user space program has to synchronize its operation on the buffer to the driver using additional read and write calls as mentioned above.

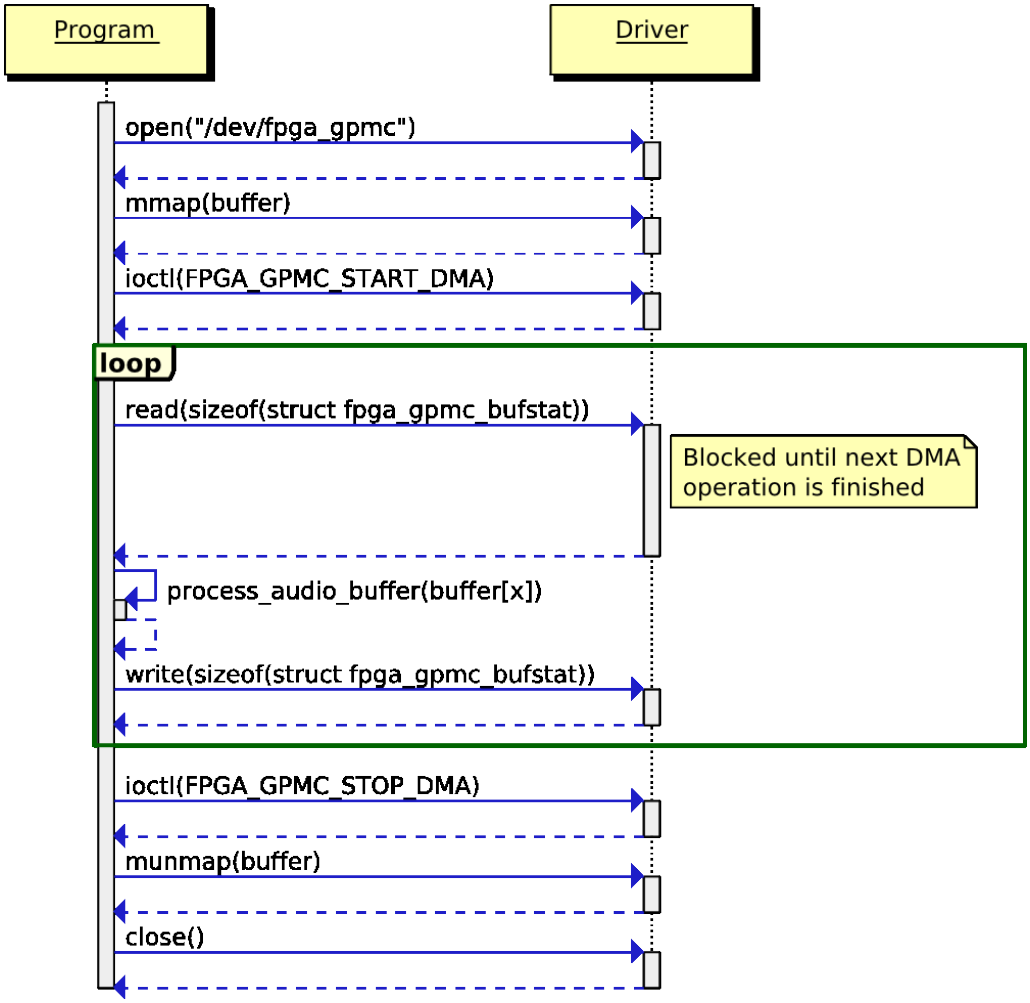

Figure 19: FPGA communication scheme

Figure 19 shows a typical use of the driver by a user space program: First the program opens the character device of the driver and maps the audio buffer memory into its address space. After that it may start to DMA transfer from FPGA with an ioctl call. In a processing loop the program can than read the audio buffer status from the driver, process received audio data from the previously memory mapped audio buffer and after that has to write the current buffer status back. If the program exits its processing loop it has to stop DMA transfer with an ioctl, unmap buffer memory and at last close the character device file.

4.3 APU Daemon

The APU Daemon is responsible for the integration of the individual APUs into the occupancy sensor network. It registers the APU with the BEMO server respectively the APU Gateway. On the APU side, the Daemon provides a Dbus interface for other software components generating sensor data which has to be transmitted.

4.3.1 Network connectivity

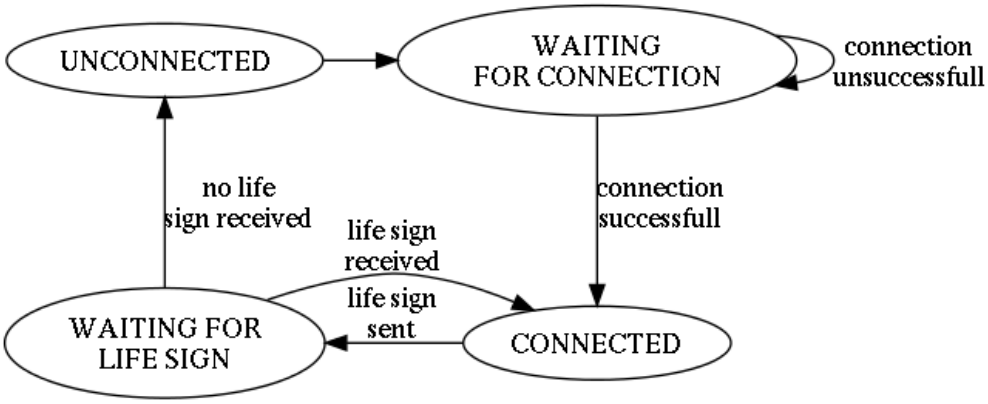

Figure 20: APU network connection state machine

Figure 20 shows a state machine of the APU Daemons network connection functionality. After startup the daemon tries to establish a connection to the APU Gateway (IP address of the BEMO server has to be supplied as a command line parameter, see next section). This connection is SSL encrypted by default, which can be disabled by command line parameter. If the connection attempt was successful the Daemon enters the CONNECTED state and regularly sends a life sign message to the APU Gateway to ensure the connection still exists and is active. If there is no reply to the life sign message from the Gateway within a timeout, the Daemon closes the connection and enters UNCONNECTED state again.

4.3.2 Usage

The APU Daemons binary can take several command line parameters, listed below:

- help: Display the parameter list with short explanations.
- host: Specifies the BEMO servers IP address and port (for example: --host=192.168.0.10:6789).
- logfile: Enables logging to a file in addition to standard output.
- loglevel: Sets the level for logging messages from 0 (all log messages) to 5 (no messages). The level names in detail from 0 to 5 are: trace, debug, info, warn, error and fatal.
- ssl: Enables or disables (options: on or off) SSL encryption of the network traffic between APU Gateway and APU Daemon. Daemon and Gateway have to use the same setting of course. Default setting is "on".

The Daemon is usually started automatically at APU boot up through systemd. Network settings (IP address of the BEMO server, etc.) are taken from the APU's network configuration stored in EEPROM (Section 6.3.2). Starting and stopping the Daemon should be accomplished using systemd's control program (Section 6.3.3).

4.4 ADATReader

The AdatReader provides a framework for the processing of audio data captured by the ASU units connected to the APU. It uses the GPMC device drivers character interface (Section 4.2.3) to get audio data and status information and connects to the APU's system Dbus for transmitting sensor data and detected events to the APU Daemon. The actual audio processing is done by plugins.

4.4.1 Plug-in interface

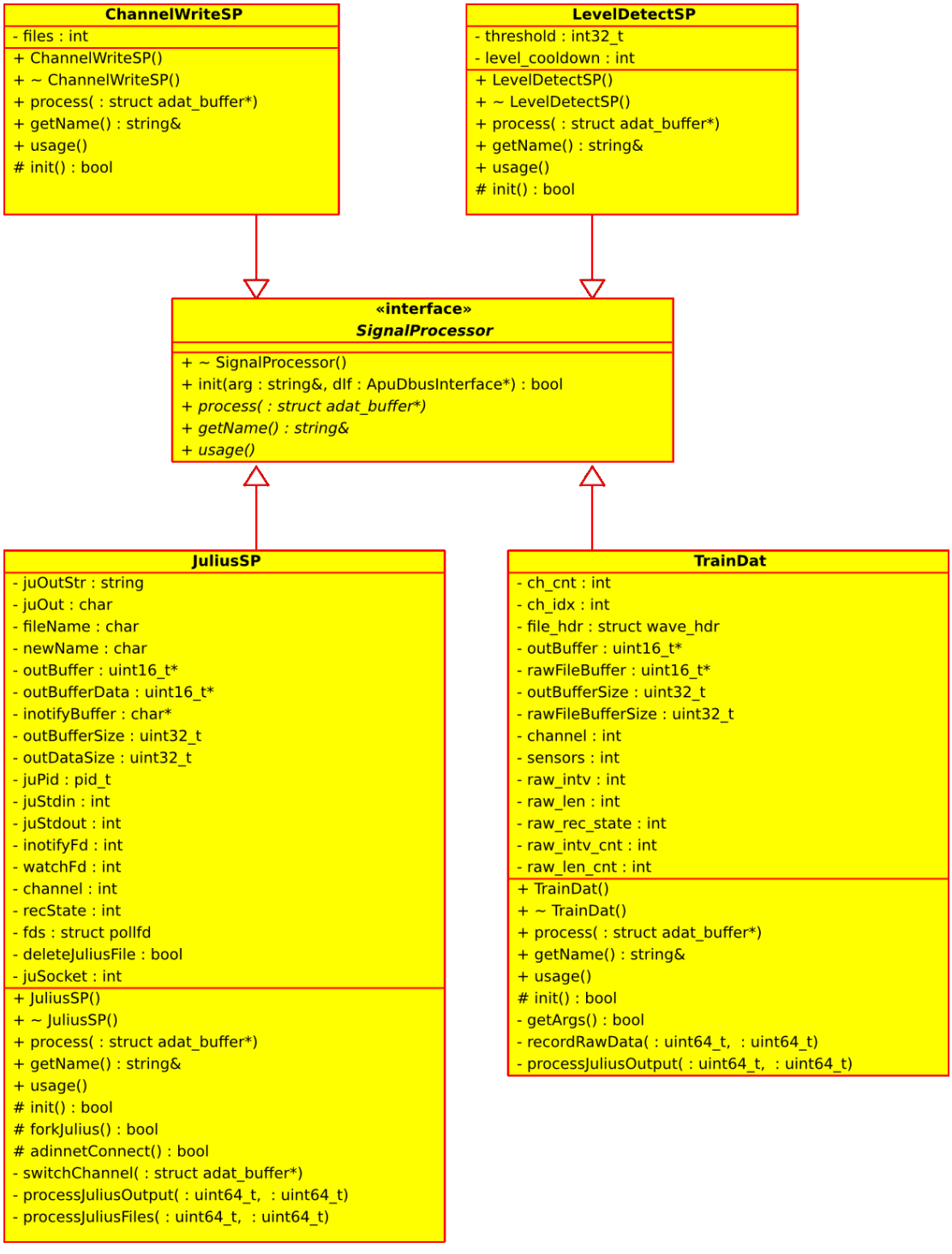

Figure 21: APU plugin interface structure

To allow a modular design of the audio processing algorithms and to support individual development efforts the AdatReader provides a plugin interface for audio data processing. Each plugin is integrated into the audio data processing chain of the AdatReader. As input every received audio buffer is supplied to the plugin, which may then process this data, apply any changes and/or deliver results of its computations (for example sensor values) via the Dbus to the APU Daemon. The plugin interface is realized using the Qt framework plugin API and provides a C++ interface class. Figure 21 shows this interface class (SignalProcessor) and 4 plugins implementing this interface. Methods in the interface declaration, which have to be implemented by each plugin are:

- init(...): This method is called upon plugin initialization providing an argument string and a reference to the Dbus interface class.
- process(...): This method gets called for every received audio buffer and provides a pointer to this audio data. The plugin should implement its actual processing work in this method.
- getName(...): This method has to return the plugins name.
- usage(...): This method should print an argument description of the plugin to stdout.

4.4.2 Available plugins

The following individual plugins are available in the default AdatReader installation on the APU:

- ch_write: This plugin allows recording the audio data to individual files per channel. This is mainly used for test purposes and to verify the signal processing chain (for example to compare audio data before and after a signal processing plugin).
- julius: This plugin is used for occupancy detection by means of the Julius speech recognition framework. An instance of the Julius framework is created by the plugin. Input audio data is delivered using the adinet socket interface of Julius. Sensor values (occupancy levels) are obtained by parsing the output of the Julius process.
- level: The level detect plugin recognizes if a configurable level threshold for each individual channel is reached. This information is used for the microphone activity display in the APU Gateway (Section 6.1.4).
- traindat: With this plugin data for (re)training purposes can be gathered . It allows transmission of audio data in configurable time slices to the BEMO server.

4.4.3 Usage

The AdatReader supports the following command line parameters:

- -f/--log-file: Enables logging to a file in addition to standard output.
- -l/--log-level: Sets the logging level (possible values are ERROR, WARNING, INFO, DEBUG, DEBUG1, DEBUG2, DEBUG3, DEBUG4).
- -s/--sp-list: Adds a plugin with its parameters to the signal processing chain. This Parameter can be supplied several times (for each plugin which should be used). Plugins are added to the signal processor chain in order of their occurrence in the command line.

Plugin parameters are supplied directly after the plugin name separated by a colon. The following parameters are supported by the individual plugins:

- ch_write:[path]
  - path: Path to a directory where the individual channel audio data files should be stored (if omitted: /tmp).
- julius:[conf:rt]
  - conf: Configuration file which should be used for the Julius speech recognition framework. The working directory for Julius is also determined by this file.
  - rt: Record time in seconds for each channel.
- level[:threshold]
  - Threshold value to use for level detection. A positive 23 bit integer value has to be supplied.
- traindat[:ch_mask:intv:len]
  - ch_mask: Hex encoded 24 bit binary mask of channels to record raw audio data from. For example a value of 3 means to record channel 0 and 1. A value of 800C means to record from channels 2, 3 and 23.
  - intv: Interval between recordings in seconds.
  - len: Length of each recording in seconds.

For example to start the AdatReader with occupancy detection by the julius plugin and additionally the level detect plugin activated with a threshold of 50000 this command line can be used:

```
# ./adatReader -s julius -s level:50000
```

Normally the AdatReader process should be started using the respective systemd service (Section 6.3.3). The configuration of this instance started through systemd is supplied by the wrapper script: /opt/s4ecob/bin/adatReader.sh. Configuration changes can be achieved by changes in this script.

5 TECHNICAL SYSTEM VALIDATION REPORT

5.1 Introduction

In Deliverable D2.3 (Section 5) the validation methods and tests of the technical components in the S4ECoB system, which have to be performed to ensure a proper working system, are described. Following the results of the technical system validation are reported.

5.2 Test settings and results

5.2.1 APU communication test

The basic communication, receiving and encoding of the ADAT protocol and transmission to the APU processor was tested using a PCI ADAT card installed in a PC. Figure 22 shows the test setup.

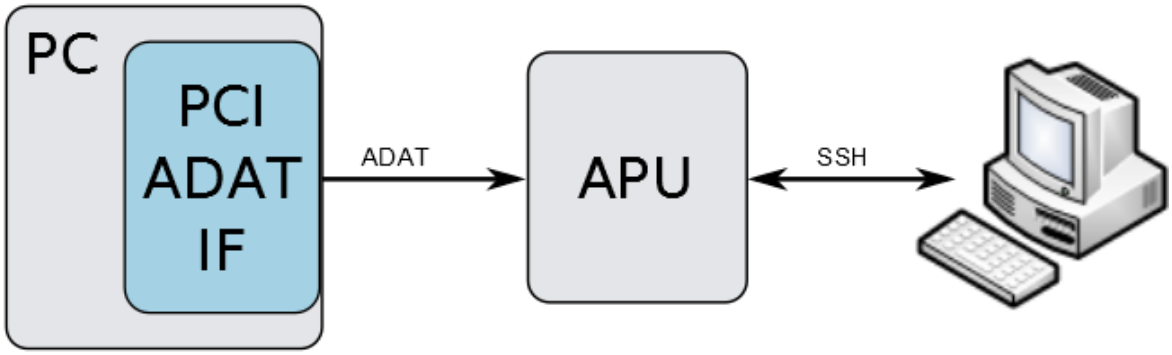

Figure 22: APU communication test setup

Several signals (Ramp, Square and sinusoidal) were generated, transmitted from the PCI card to the APU, decoded in the FPGA, transferred to the APU processor and recorded using the ch_write plugin (Section 4.4.2). A remote SSH connection from a PC to the APU was used to monitor and analyze the recorded data.

All signals could be transmitted successfully. So the audio data receiver in the FPGA and the communication between FPGA and processor were verified.

5.2.2 Audio sensor network communication test

In this test the communication in the audio sensor network (between ASU and APU) was verified. The cycle to cycle and period jitter of the sample clock was measured to ensure proper function of the ASU.

The data transfer from ASU to APU using the ADAT protocol was already measured and documented in Deliverable D3.1 (Section 5.6). The noise and total harmonic distortion measurements of the data transmission is documented in Deliverable D3.1 (Section 5.4 and 5.5) and in a test with 40 m cable

length between APU and ASU no communication errors could be detected. The ADAT receiver was already verified in the APU communication test.

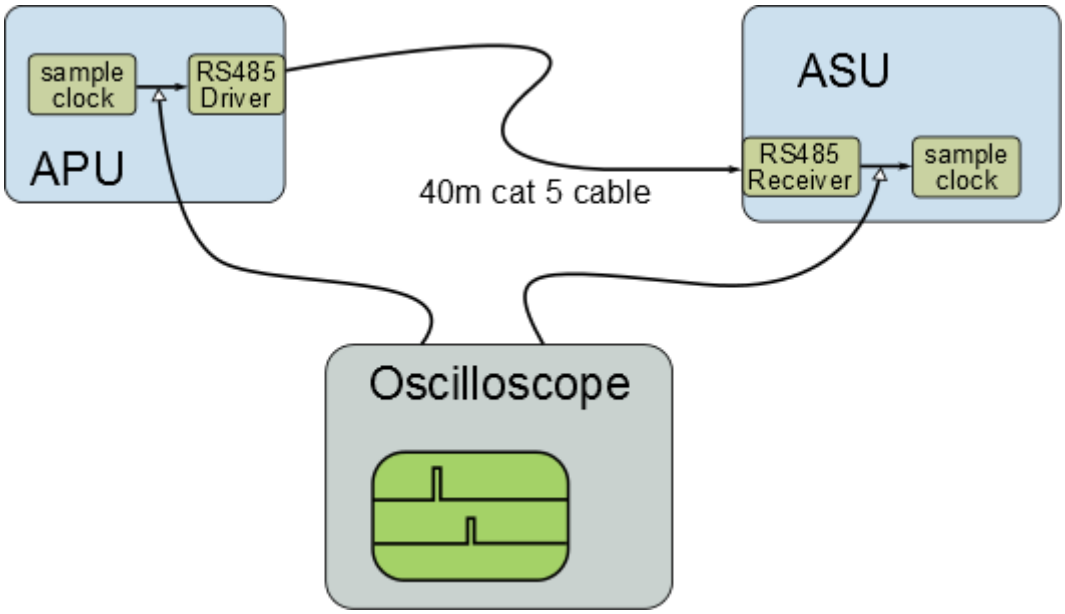

Figure 23: Audio sensor communication measurement setup

In Figure 23 the test setup is shown. The sample clock (48 kHz) is provided by a clock generator IC (PLL1707) on the expansion board of the APU and is distributed through the FPGA to the RS485 Driver ICs and then transmitted over cat 5 cable to the ASU. A RS485 receiver ICs converts the signal to TTL level in the ASU. The clock at the input of the RS485 driver in the APU and the clock at the output of the receiver in the ASU were measured using two channels of a digital oscilloscope.

- Cycle to cycle jitter

  Cycle to cycle (C2C) jitter is defined in JEDEC Standard 65B as the variation in cycle time of a signal between adjacent cycles, over a random sample of adjacent cycle pairs [8].

  The result of the measurement, a oscilloscope screenshot of the sample clock at the output of the RS485 receiver in the ASU are shown in in Figure 24. Within over one million cycles no jitter was measured. A deviation from the 48 kHz sample frequency was not detected.

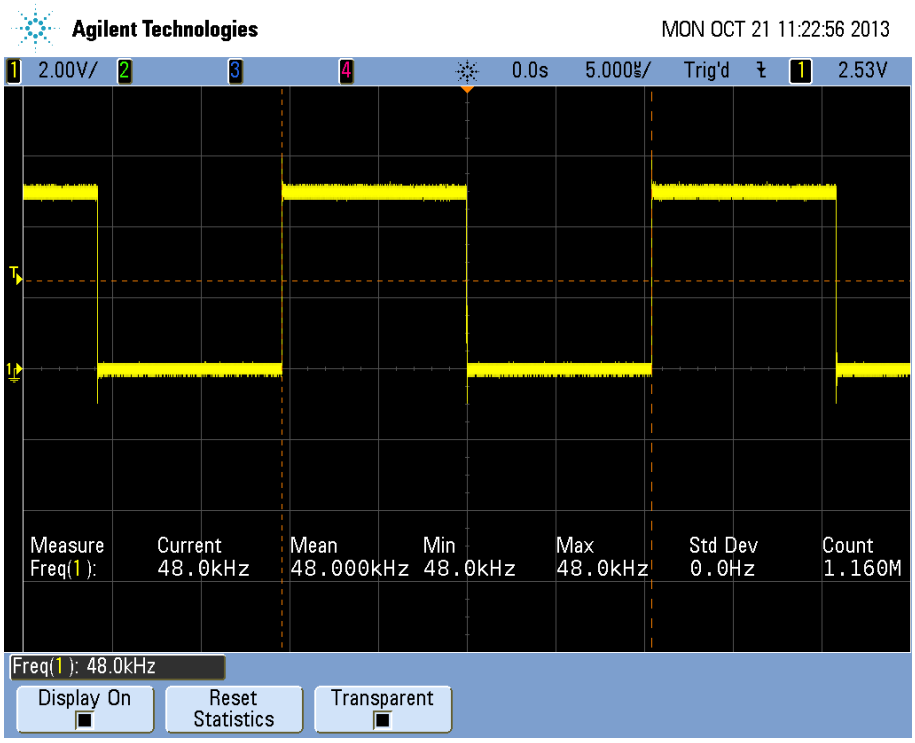

Figure 24: ASU cycle to cycle jitter measurement

- Period jitter

Period jitter is the deviation in cycle time a clock signal with respect to the ideal period over a number of randomly selected cycles.

In Figure 25 the screenshot of the oscilloscope with the measurement result is shown. The first channel (yellow line) shows the sample clock in the APU and the second channel (green line) the clock in the ASU. The following delays were measured:

- 234.5 ns minimum delay
- 239.5 ns maximum delay
- 236.78 ns mean delay
- 1.0361 ns standard deviation

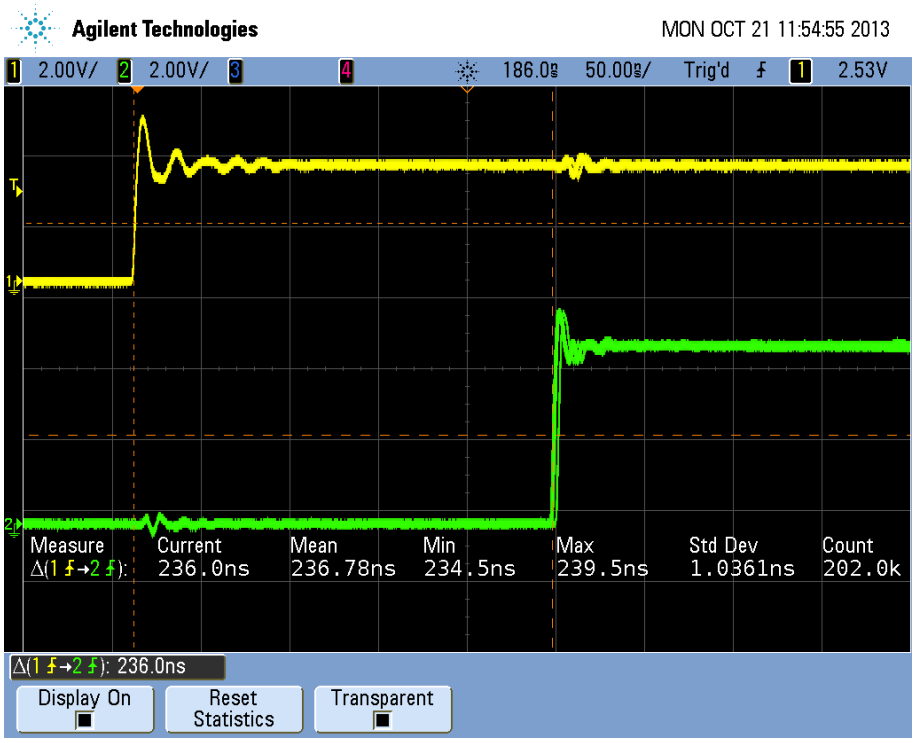

Figure 25: ASU phase jitter measurement

The period of the sample clock (48 kHz) is 20,833 µs, so a jitter in the sample clock in the ns range is negligible and has no effect in the audio signal quality. The result of the jitter and the noise and distortion measurements is that a 40m cable could be used to connect the APU and the ASUs with no influence on the signal quality.

5.2.3 Time synchronization

Time between APUs and BEMO server is synchronized using the precision time protocol (PTP). Tests of the precision of this synchronization are already described in Deliverable D2.3. In the test described in the following, the time until all APUs and the BEMO server are synchronized is measured, as it adds to the startup time of the complete network because of occupancy values identified by the network are only valid after PTP synchronization has finished.

For these tests two values were measured: The time needed for the PTP daemon on the APU to synchronize after connection to the APU Gateway was established and the time PTP daemon and server needing to synchronize if the PTP server starts after APU connection to the Gateway is already established.

Average time until synchronization after APU connect	2s
Average time until synchronization after PTP server restart	15s

Figure 26 shows the APU Gateway's graphical user interface displaying 7 connected APUs which are time synchronized to the BEMO server (indicated by a green clock symbol for each APU).

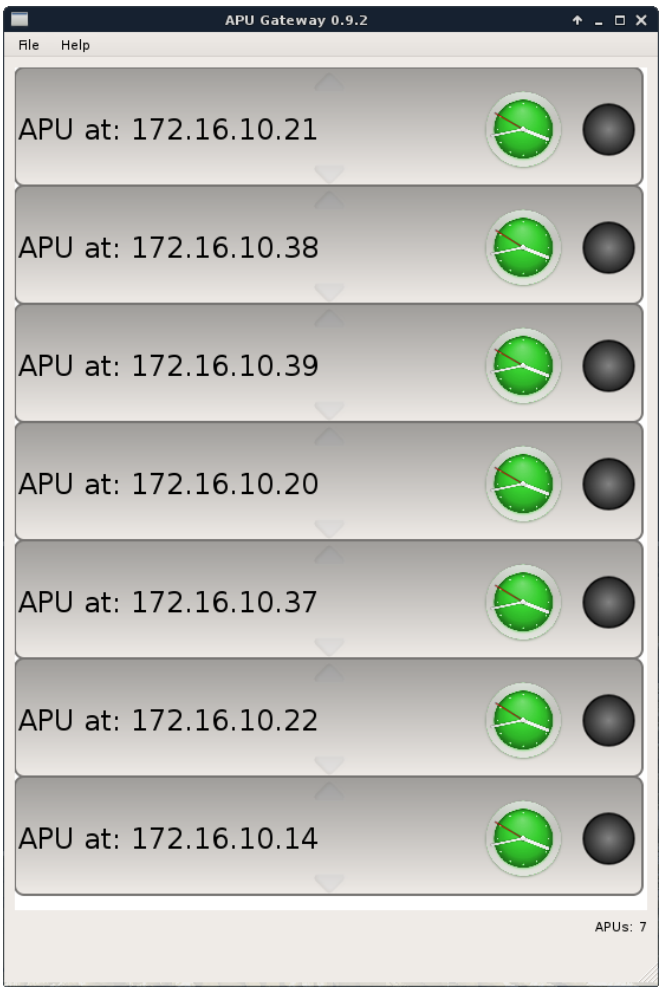

Figure 26: APU Gateway screenshot

5.2.4 Sensor signal propagation delay

In this test the possible reaction time of the occupancy sensor network to sound events is measured and classified. This signal propagation time (not necessarily essential for occupancy detection but for extended uses of the network like localization) between the occurrence of a sound event up to registration of this event by the BEMO server is limited in several ways:

- Buffering of the audio data stream in the APU's FPGA and CPU's memory
- Signal processing algorithms of the APU
- Delays introduced by the network connection and TCP streams SSL encryption between APU and BEMO server
- Receiving and processing of the network messages by the APU gateway

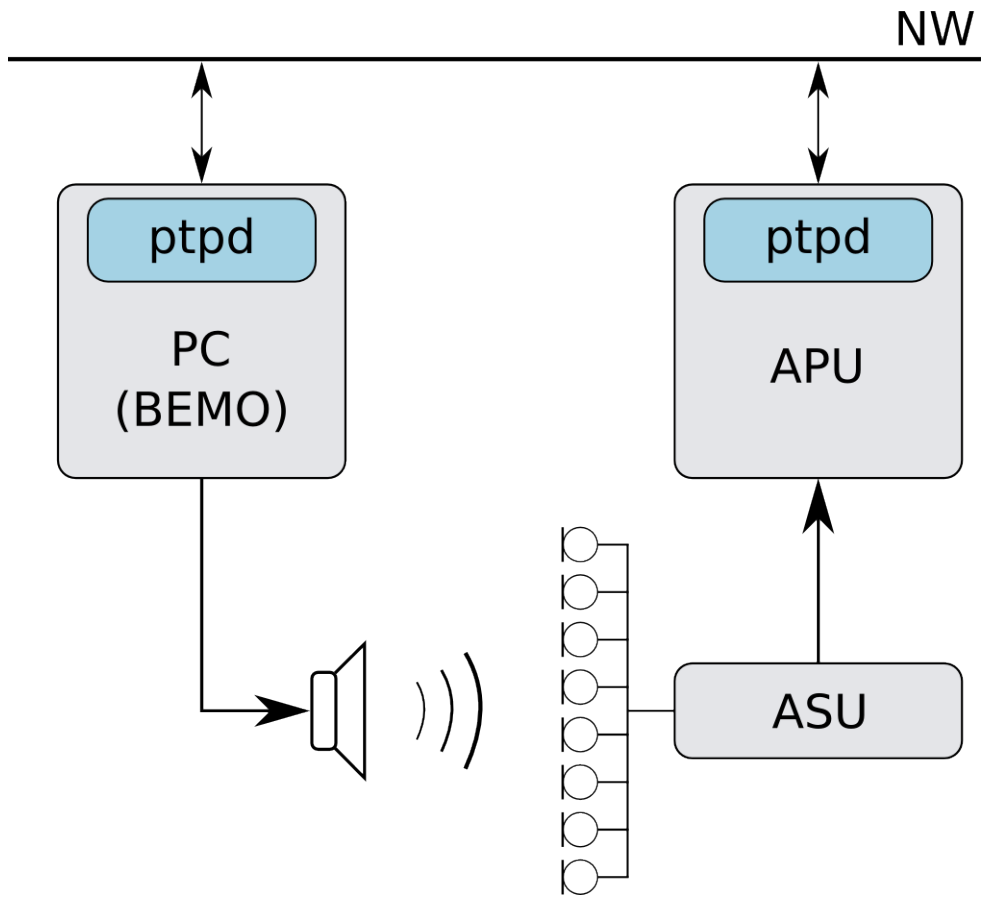

Figure 27: Signal propagation measurement setup

This signal propagation delay was measured using the test setup shown in Figure 27. A PC (in the role of the BEMO server) and an APU are connected via network. Local time of PC and APU are synchronized using PTP protocol (see previous section).

To perform the actual measurement, the PC generates an audio signal (1kHz sine waveform was used) by means of a connected speaker. Playing the sound was done using the aplay utility. This tool from the Advanced Linux Sound Architecture utility package allows playing audio files from the command line in various formats. To record the exact start time of audio signal a patch was applied to this tool. On the first write event of audio data to the sound hardware of the PC a time stamp is taken and printed to the command line for logging. This ensures maximum precision achievable in software of the exact time the audio signal is generated. Nevertheless this introduces a delay between first audio buffer write operation and actual tone generation of the speaker by the PC sound hardware (for example the time to fill hardware buffers) of unknown length. For this test this delay is assumed to be constant and minor in comparison to the total signal propagation delay.

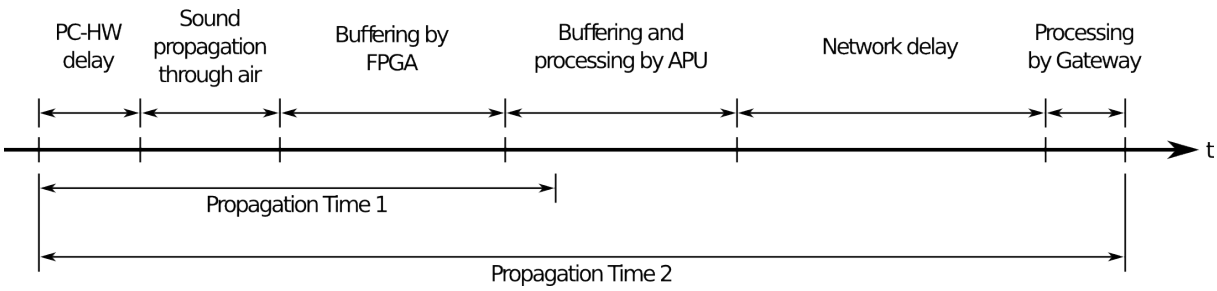

Figure 28: Signal propagation time scheme

Speaker and microphone array of the ASU are placed in a distance of 1m to each other. So after approximately 3ms the tone generated by the speaker is received and converted into an electrical signal. All 8 microphone signals are converted to a digital ADAT stream and sent to the APU by the ASU. Here the audio data is unpacked and transferred to main memory of the CPU. Each block of audio data gets marked with a time stamp after reception. In the next step signal processing by the APU is performed. For this test a simple level detection algorithm gets inserted in the signal processing chain. If a preconfigured threshold of all incoming 8 audio channels is reached, a message is generated and transmitted over the network connection to the APU gateway. Upon reception of such a message, the Gateway generates a third time stamp.

Figure 28 shows the described actions on a time line (distances are not true to scale). With propagation time 1 and 2 the delays between the recordings of the time stamps is indicated. The measurement was carried out 1000 times.

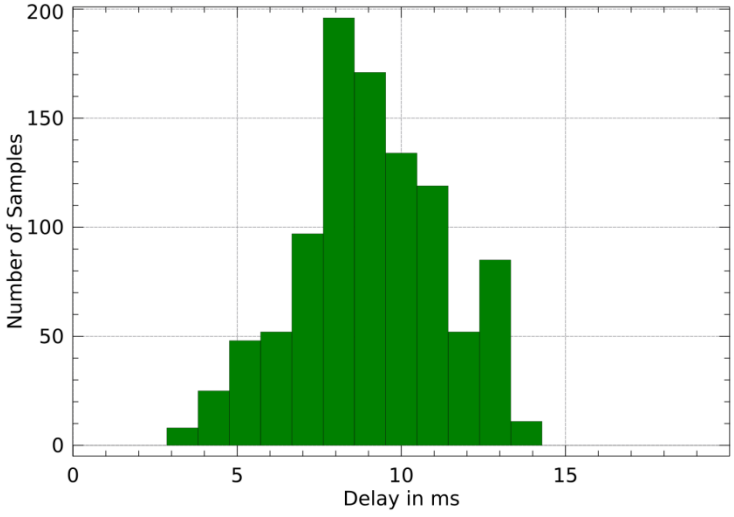

Figure 29: Signal propagation time 1

Figure 29 shows the propagation time 1 in form of a histogram with a near normal distribution around 8ms. With the minimal signal delay at 3ms and maximum at 14ms this correlates with the expected results following the delay of the sound wave between speaker and microphone (3ms) limited by the speed of sound through air and the maximum buffer depth of the FPGA of 512 samples (at 48 kHz sample rate this equals approximately 10ms).

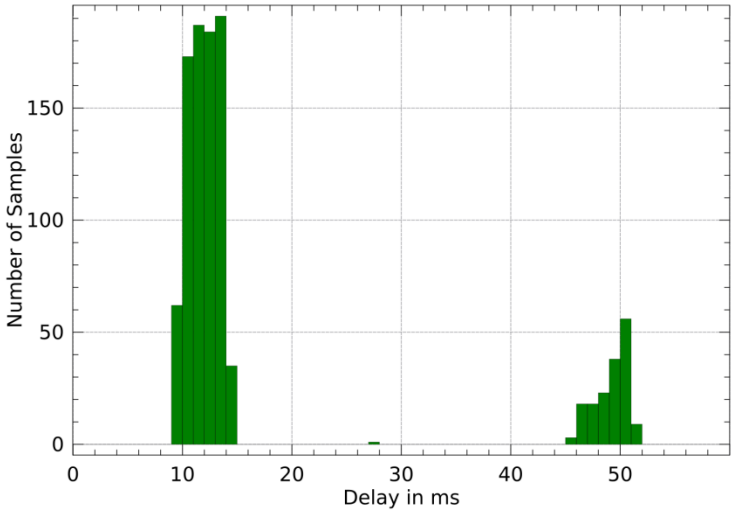

Figure 30: Signal propagation time 2

Figure 30 shows the propagation time 2 (between sound event generation and reception of the threshold message by the APU Gateway). The histogram shows the distribution of time delay at two points around 12 and 50ms. This results of additional unpredictable signal processing time of the APU by the Julius audio framework for occupancy detection and additional network activity because of other messages exchanged between APU and Gateway apart from the threshold one used for this test.

With a maximum propagation delay of 52ms and a jitter of 43ms the system was proven to be able to react fast to sound events in spite of different audio buffer mechanisms, signal processing algorithms and encrypted network based communication.

5.2.5 Remote access

Remote access to each APU is essential for monitoring, maintenance and further development of the occupancy sensor network once deployed in the demo sites. This test was carried out to ensure APU access is possible both in local network and over network boundaries using the Internet.

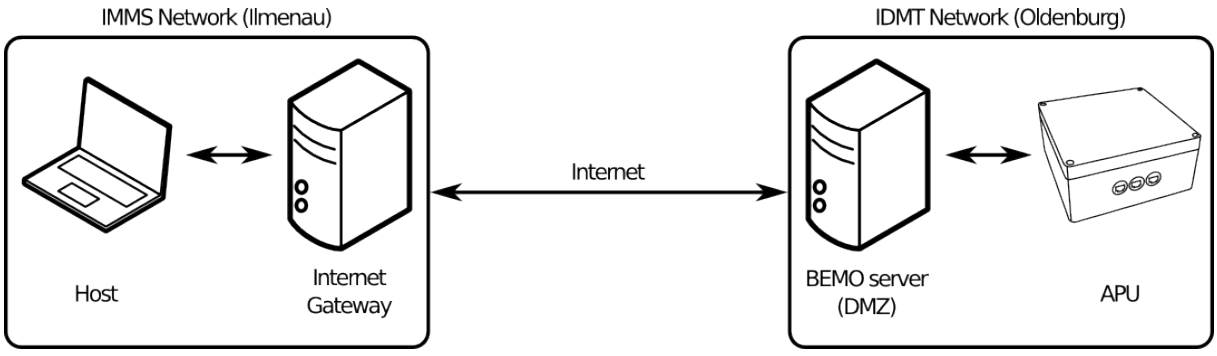

Figure 31: Remote access scheme

Tests in a local network were performed using the IMMS test network setup. Remote access was tested using the setup shown in Figure 31: An APU located at project partner FGH-IDMT in Oldenburg was accessed from IMMS Ilmenau over the Internet. The APU at IDMT is connected to a server in the DMZ which accepts connection from Internet on the secure shell port 22. The firewall at IMMS was configured to allow SSH connections from an internal host over the Internet router to the IDMT server.

Figure 32: APU remote access screenshot

Thus an SSH connection to IDMT server for the user elstts can be established with the command:

```
# ssh elstes@oldmz01.idmt.fraunhofer.de
```

After that a login to the APU with the IP address 172.16.10.14 is possible with:

```
# ssh root@172.16.10.14
```

Figure 32 shows such a login process. Using certificate based authentication and configuring the ssh client accordingly it is also possible to automate the APU login over one or more stations.

The local and remote login process was tested several times. Always a connection to the APU could be established.

Using this SSH connection updating software components or configuration data on the APU is possible.

To ensure maximum flexibility at further APU software development, updating the complete firmware image of an APU is possible, too. This process involves several steps. The first one is the already described remote login to the APU. After that, the boot mode of the APU has to be configured for rescue

boot. This is done by mounting the boot partition of the APU's interval flash card and editing the boot loader configuration file uEnv.txt:

```
root@apu:~# mount /dev/mmcblk0p1 /mnt/card

root@apu:~# vi /mnt/card/uEnv.txt
```

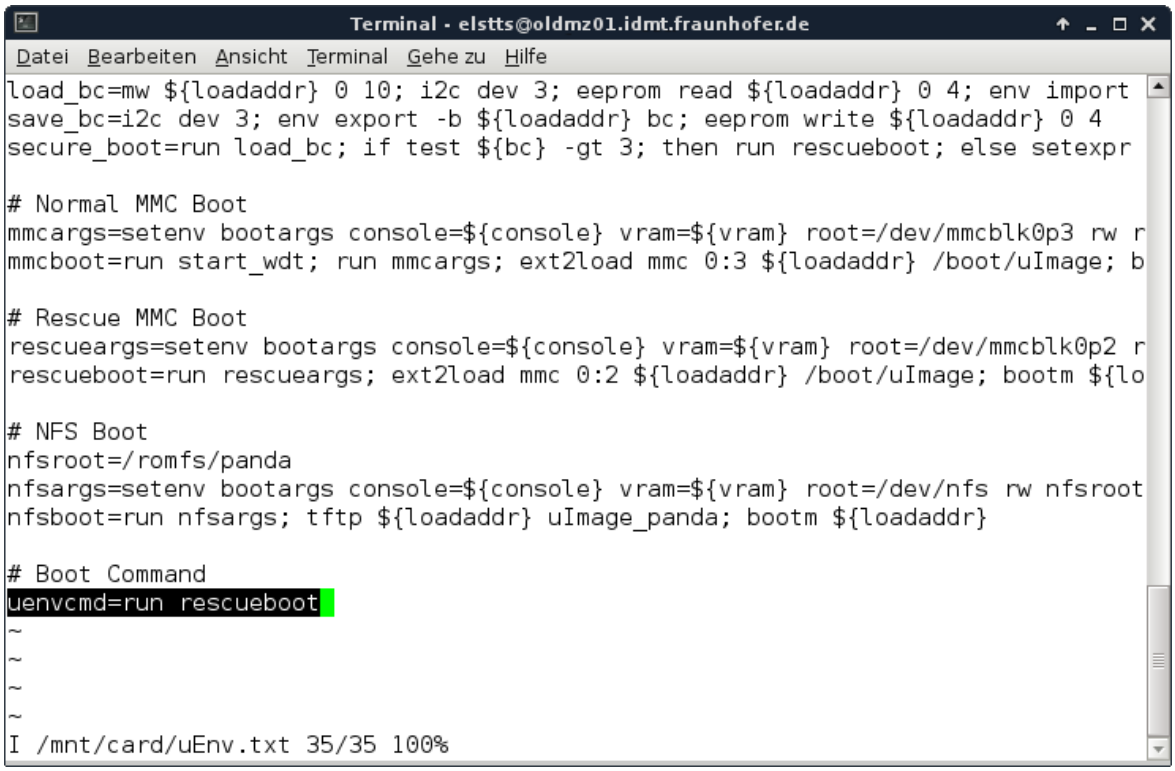

Figure 33: APU boot configuration

The variable uenvcmd at the end of the file has to be changed from "secure_boot" to "rescueboot" (Figure 33). After saving and closing the file the APU can be rebooted:

root@apu:~# reboot

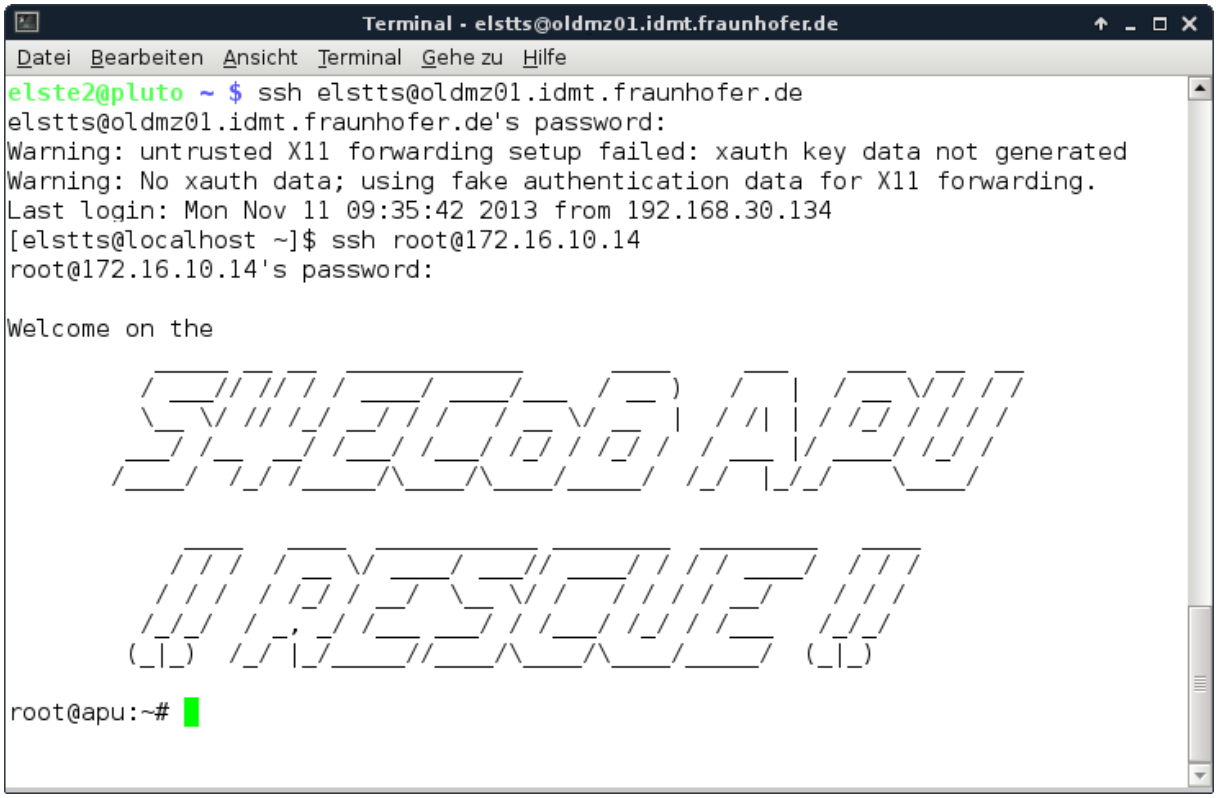

Figure 34: APU rescue login

This leads to the APU rebooting in rescue mode. The same already described login procedure can be used to re-connect to the APU via ssh (Figure 34). In the next step the partition containing the normal operating system on the flash card of the APU has to be re-formated:

```
root@apu:~# mkfs.ext3 /dev/mmcblk0p3
```

```
root@apu:~# mount /dev/mmcblk0p3 /mnt/card
```

Copying the new firmware image to this partition can be done in various ways. Avoiding an additional storage of the firmware image on the APU it can be for example extracted directly to the mounted flash card partition through an SSH pipe from the host (Figure 35).

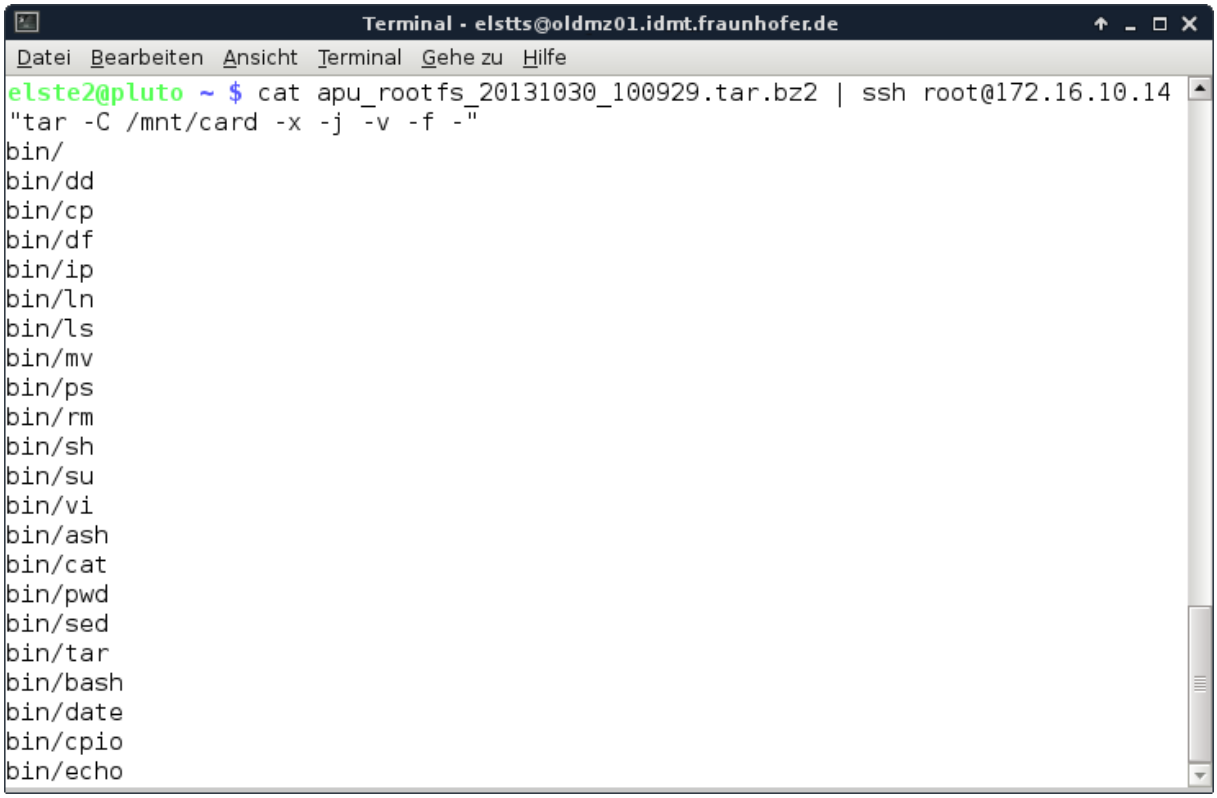

Figure 35: APU firmware copy

After un-mounting the flash card partition with the new firmware image, the boot-loader configuration has to be change again to “secure_boot”. Last step is to reboot the APU again which loads the newly copied firmware image:

```
root@apu:~# umount /mnt/card

root@apu:~# mount /dev/mmcblk0p1 /mnt/card

root@apu:~# vi /mnt/card/uEnv.txt

root@apu:~# reboot
```

This firmware update procedure has been tested several times in the local IMMS test network and remotely by IMMS with an APU located at IDMT Oldenburg. Every time the new firmware image was successfully installed.

5.2.6 Robust firmware update process

In every embedded device the firmware update process is critical as external events like for example an unexpected power loss may leave the device in an undefined state where further operations or remote access is impossible. This has to be avoided as an easy physical access to the APUs after deployment for exchanging the flash card cannot be guaranteed or is impracticable. That’s why the firmware update process of the APUs was designed to be robust.

The normal (complete) firmware update is carried out in several steps as already described in 5.2.5:

1. Remote access to APU
2. Setting boot mode to rescue boot
3. Reboot in rescue mode
4. Remote access to APU
5. Formatting flash card partition of the normal system
6. Extracting new firmware image to normal system partition
7. Setting boot mode to normal
8. Rebooting

To ensure the robustness of the firmware update process a power loss situation was simulated in this test at every of these steps. The results after the reconnection of power to the APU are listed below:

Power loss in state	Behavior
1	APU reboots in normal operation mode
2	APU reboots either in normal or rescue mode (depending of exact timing of the power loss)
3	APU reboots in rescue mode
4	APU reboots in rescue mode
5	APU reboots in rescue mode (partition has to be formatted again, to continue update)
6	APU reboots in rescue mode (partition has to be formatted and the image extracted again to continue update)
7	APU reboots in rescue mode or to new system firmware (depending of exact timing of the power loss)
8	APU reboots to new system firmware

Table 14: APU behaviour after power loss

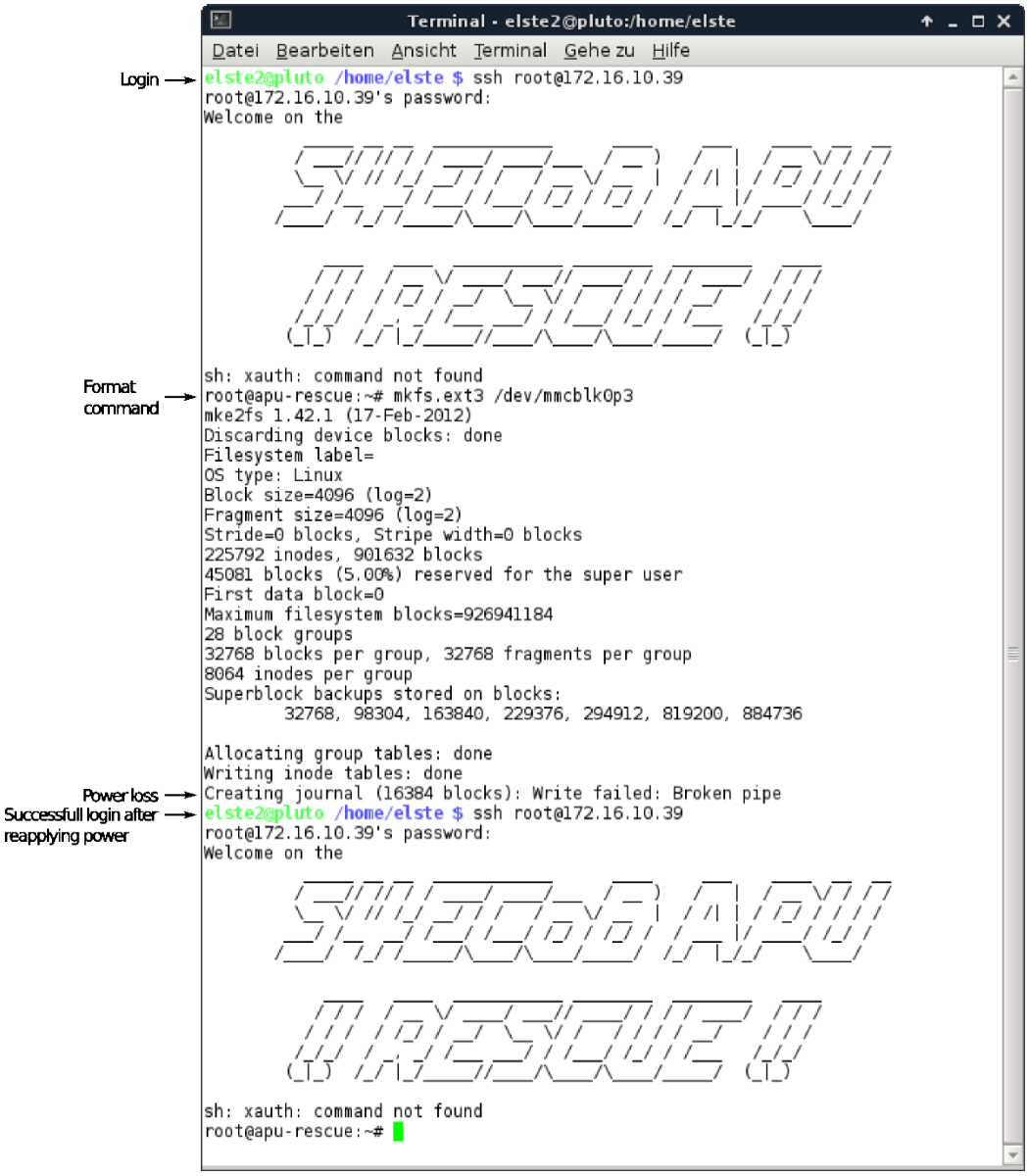

Figure 36: Screenshot of a power loss during flash card format at firmware update

Figure 36 shows a screen shot of the update process at step 5 (re-formatting the flash card partition) during a power loss, where the SSH connection gets terminated. After reapplying power to the APU login to the rescue system is possible again.

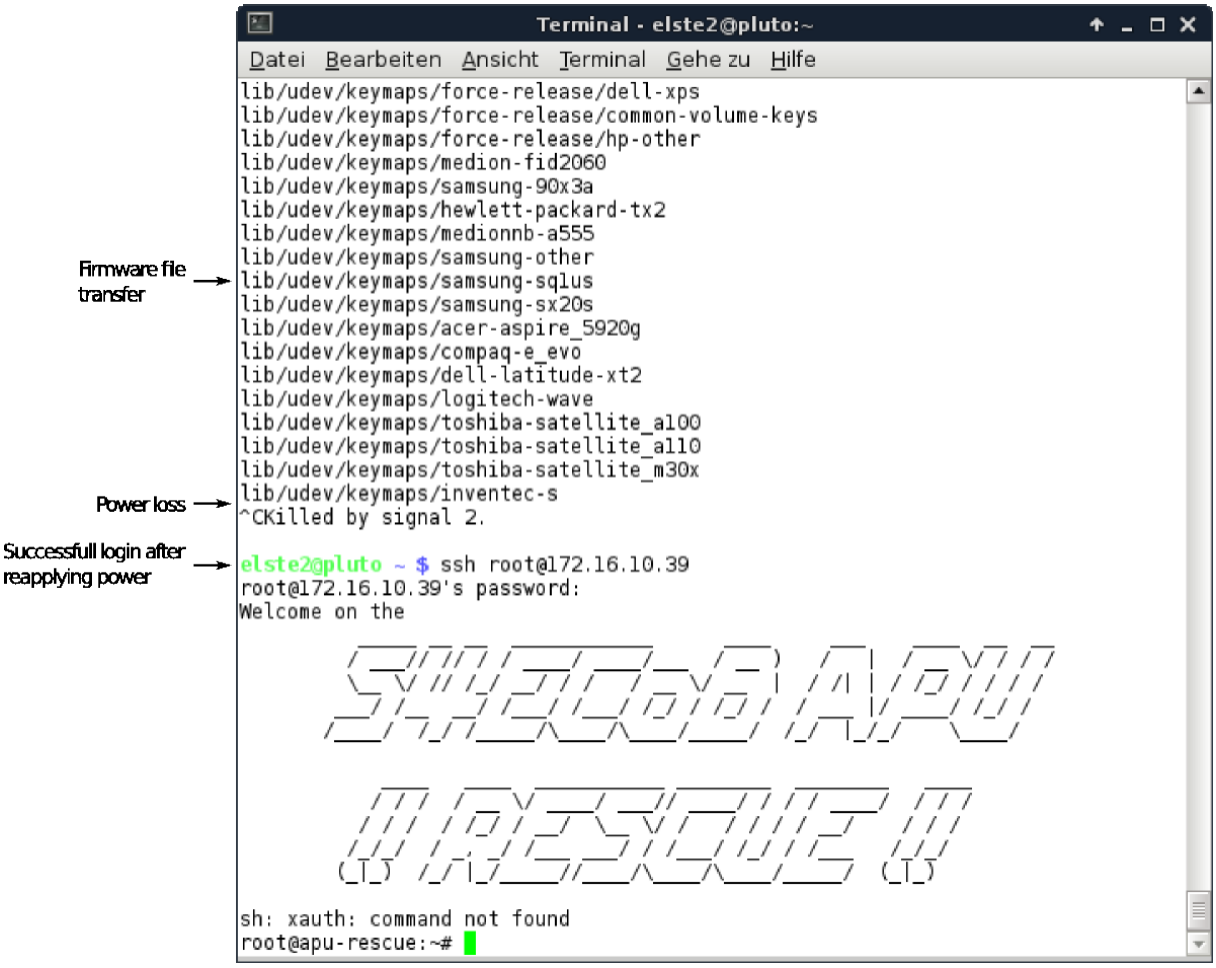

Figure 37: Screenshot of a power loss during firmware image extract at firmware update

Figure 37 shows a screen shot during a power loss a step 6 (copying of the new firmware image) of the update process. Again the copy process gets terminated but login is again successful after powering up the APU.

At no time the APU was inaccessible after regaining power supply.

5.2.7 BEMO communication

A test setup consisting of several APUs (maximum 7 APUs in different configurations) and an APU gateway running on a BEMO server was installed in the IMMS company network.

In this test network the occupancy network communication and the data transfer between APUs and APU gateway was tested. The following tests were carried out:

- Data transmission of the occupancy levels, status information from the APUs to the BEMO server
- Raw audio data transmission from APUs to BEMO server
- Encryption of the data transfer

The APUs were tested in different configurations:

- One APU with three ASUs connected
- Three APUs with each one ASU connected
- Seven APUs in the test network

In all test setups the communication between the APUs and the BEMO server was successfully verified.

5.2.8 APU network log-in

To ensure a constant operation of the occupancy sensor network it has to be robust against network errors where the connection between APU and BEMO server is separated (either physically, through software filters (for example a misconfigured firewall) or power loss of BEMO server or APU). After reestablishment of the network connection the data channel between APU and APU Gateway should be automatically restored as well.

Therefore the time between initial boot of the APU until registering with the APU Gateway, the time until the APU reconnects to the Gateway after a network error and the time upon starting the APU Gateway until connection with an already running APU (simulating a restarted BEMO server) has been measured. The results are listed below.

Average time until connection after APU reboot: 56s

Average time until connection after APU network error: 53s

Average time until connection after APU Gateway start: 3s

This tests shows, that even after a complete power loss the network is operational again in less than one minute.

5.2.9 Environmental conditions and power consumption

5.2.9.1 Power consumption

The power consumption of the APU was measured using the N6705B DC power analyzer from Agilent Technologies. The input voltage of the APU (5V) was supplied by the power analyzer and measured.

Figure 38 shows a screenshot of the power consumption measurement of one APU with no ASU connected. In this state only the network components, the time synchronization and communication with the APU gateway are running, thus an average power consumption of 2.65W was measured.

Figure 38: APU power consumption with no ASU connected

Figure 39 shows a screenshot of the power measurement with three ASUs connected to the APU. In this measurement setting the AdatReader with the Julius (Section 4.4.2) plugin running on the APU and the audio data from all three ASUs were processed in the APU. In this setup, that is used in the pilot sites the APU requires an average power of 2.85 W and a maximum power of 3.44 W.

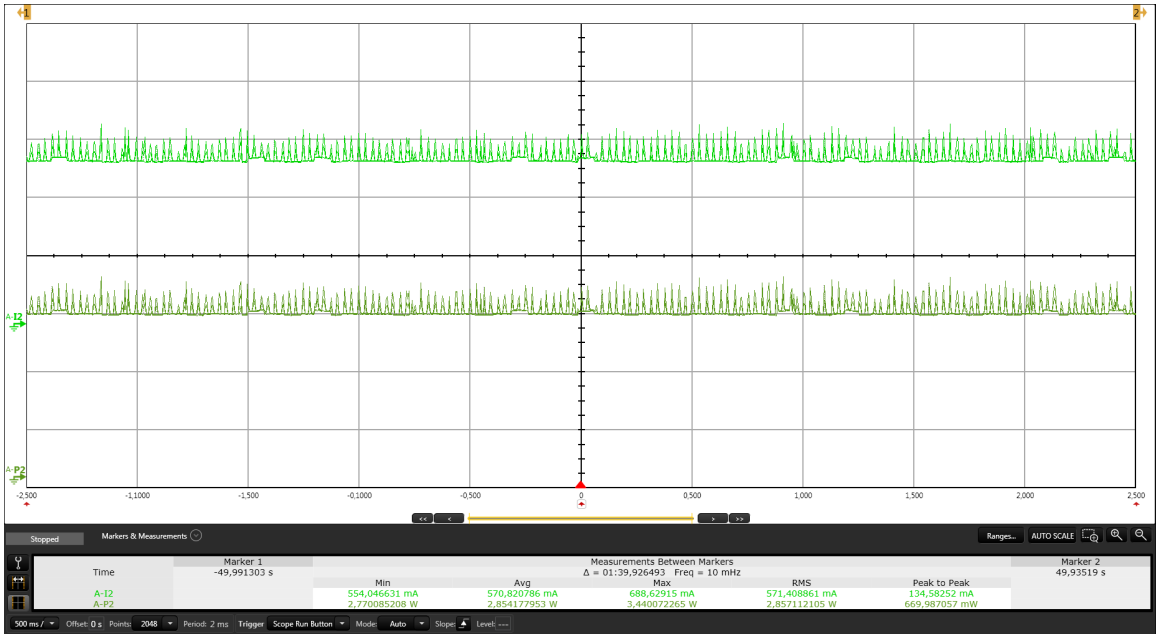

Figure 39: APU power consumption with three ASU connected

In Figure 40 an enlarged detail of the previous measurement is shown. The time between the two markers (power spikes) is 5.37ms. This time corresponds to the FPGA read access cycle, 256 audio data values were stored and then transferred to the processor.

$$t = 256 * 1/48kHz = 5.333ms$$

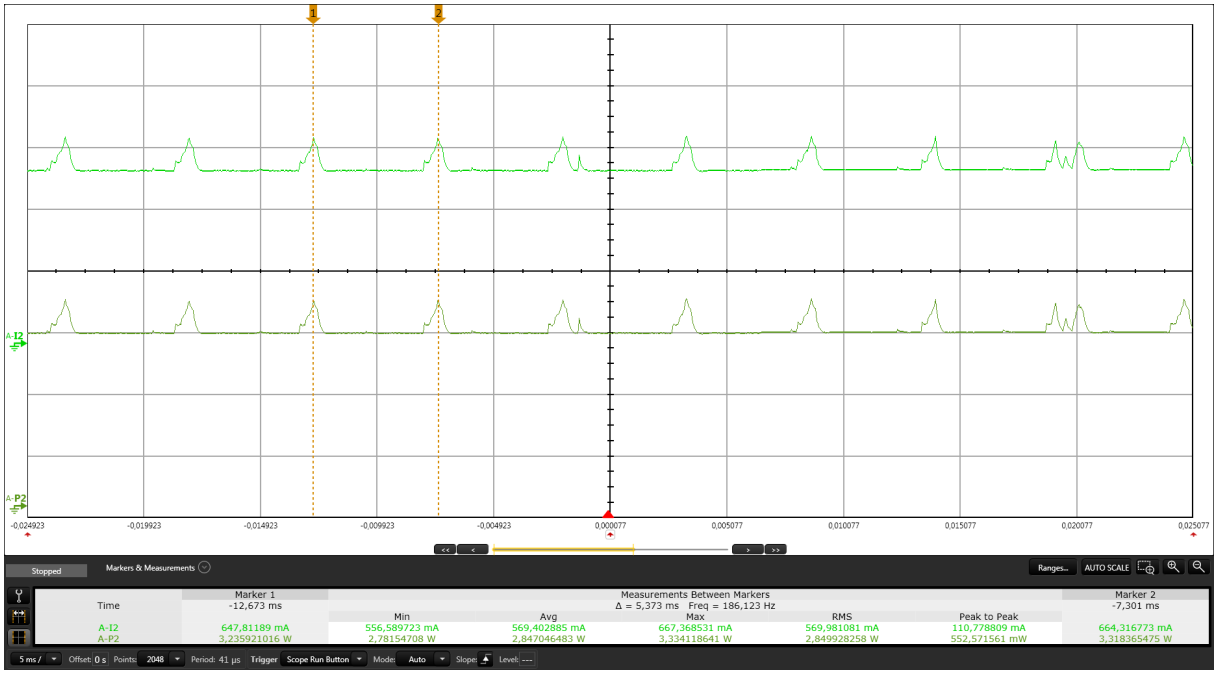

Figure 40: APU power consumption GPMC access

These measurements show that the APU is an energy efficient platform. A common personal computer consumes between 30 and 100 W, compared with this the APU needs less than 10% of the electrical power.

5.2.9.2 Environmental Conditions

The environmental test was performed to ensure the proper work of the APU under different environmental conditions. In this test setup the APU was put inside a conditioning cabinet (Figure 41), one ASU and a Laptop, acting as BEMO server was connected to the APU.

Figure 41: APU in conditioning cabinet

The following tests were carried out:

- APU heartbeat

Check APU life sign in the APU gateway, so occupancy network connection is still working.

- APU reconnect

Disconnect the APU from the BEMO server and reconnect after a while. Check if the APU establish the connection with the APU gateway.

- ASU reconnect

Disconnect the ASU from the APU and check status change in the APU gateway. Reconnect the ASU after a while and check status change in the gateway.

- Data transfer

The successful data transfer from the ASU to the APU and to the APU gateway was checked using the level plugin running on the APU (see 4.4.2) by generating noise in front of the microphones and check the response in the APU gateway.

Table 15 lists the test results under the different environmental conditions.

Condition	Test	Result
50°C for 2 hours		
	APU heartbeat	passed
	APU reconnect	passed
	ASU reconnect	passed
	Data transfer	passed
0°C for 2 hours		
	APU heartbeat	passed
	APU reconnect	passed
	ASU reconnect	passed
	Data transfer	passed
50°C for 2 hours		
	APU heartbeat	passed
	APU reconnect	passed
	ASU reconnect	passed
	Data transfer	passed

Table 15: APU condition test results

Under all conditions the tests were successfully performed and the APU worked as expected. No errors or unexpected behaviour was detected.

5.2.10 Long-term test

As described in Section 5.2.7 a test setup consisting of several APUs and an APU gateway running on a BEMO server was installed in the IMMS company network. The test network was operating over a period of 3 months, the log files generated by the APU gateway were analyzed and no failure or unexpected behavior was noticed.

5.3 Test conclusions

Table 16 shows the hardware and software requirements for the embedded acoustic processing unit as defined in Deliverable D2.3 and the summarized results of the technical system validation.

The requirements R_A 1.4, R_A 4.1, R_F 4.3, R_A 4.4 and R_N 5.1 cannot be verified in a test setup.

The architectural requirement R_A 1.4 – "Easy to use and install in the demo sites" was considered during the whole APU development process. For all wired connections standard cables can be used. In the commissioning manual (Annex B) the installation of an APU is described. In general it follows the user-friendly plug-and-play paradigm. The software architectural requirements R_A 4.1 – "Use of open-source software for APU wherever possible" and R_A 4.4 – "Secure network communication and APU access restrictions" have been considered as part of the overall software architecture and its implementation.

The functional requirement R_F 4.3 – "Secure remote APU access for audio algorithm and APU firmware update using Internet and demo site specific networks (including firewalls and VPNs)" have to be tested as soon as the APUs are installed in the demo sites and a remote access to the BEMO server is available.

The requirement R_N 5.1 – "Electrical safe power supply for APU" is addressed by using a standard wall plug power supply for the APU.

In summary, all APU requirements as defined in Deliverable D2.3 were successfully checked and verified.

test # / Req.	1	2	3	4	5	6	7	8	9	10
5.3.1 R_F	x	x								
5.3.2 R_F		x								
5.3.3 R_A		x								
5.3.4 R_A										
5.3.5 R_A							x			
5.3.6 R_A							x			
5.3.7 R_F					x					
5.3.8 R_F								x		
5.3.9 R_F				x						
5.3.10 R_F				x						
5.3.11 R_N									x	
5.3.12 R_A		x					x			
5.3.13 R_A										
5.3.14 R_F			x							
5.3.15 R_F										
5.3.16 R_A										
5.3.17 R_N										
5.3.18 R_N									x	
5.3.19 R_F										x
5.3.20 R_F						x				

Table 16: Test and requirements coverage

6 APU AND OCCUPANCY SENSOR NETWORK SETUP AND MANAGEMENT MANUAL

In this section the overall setup and management of the occupancy sensor network will be explained. This includes the necessary setup of the APU Gateway and PTP daemon on the BEMO server as well as the remote management of the APUs in case of any updates or necessary configuration changes. It is mainly targeted at engineering and IT personnel. Physical installation and commissioning of the APUs is covered separately in Annex B.

6.1 APU Gateway

6.1.1 Installation

The APU-Gateway can be provided as GUI and non-GUI version for i686 and x86_64 architecture and 2 different package management systems (RPM Package Manager, Debian Package Manager) or as binary archive. When using an RPM- or DEB-based Linux distribution the Gateway packages can be installed using the distributions package manager application.

For example using Ubuntu (13.04):

```
# sudo dpkg -i apuGateway[Gui]-<version>-<arch>.deb
```

CentOs (6.4) requires the addition of 2 software repositories prior to installing the Gateway as it depends upon the Qt Framework >=4.7 and the Protobuf library (see next section), both not available in the default CentOs repositories:

```
# su
# yum install http://ftp-stud.hs-esslingen.de/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
# yum install http://software.freivald.com/el/6/x86_64/os/software.freivald.com-2.0.0-0.el.noarch.rpm
# yum update
# yum install apuGateway[Gui]-<version>-<arch>.rpm
```

On Linux distributions without RPM or DEB support a binary package archive can be provided. This archive has to be extracted. After that the Gateway can be started directly from the generated directory. All library dependencies (see next section) have to be resolved manually of course.

```
# tar xjf apuGateway[Gui]-<version>-<arch>.tar.bz2
# cd apuGateway[Gui]-<version>-<arch>/bin
```

```
# ./apuGateway[Gui]
```

6.1.2 Dependencies

Runtime dependencies for the APU Gateway are the Qt Framework libraries >=4.7, the Protobuf library (>=2.4.0) and a running Dbus daemon.

6.1.3 Usage

After installing the Gateway it may be called from a terminal by typing "apuGatway" or "apuGatewayGui" respectively for the GUI version. The Gateway can be called with several parameters. A full list of the available options are printed to the terminal by calling the Gateway with "--help". Parameters and values follow the GNU long option syntax, meaning parameters are started with "--" and separated from values by "=". For examples setting the Gateways network port to 7000 and the audio data output directory to /tmp is done by calling: "apuGateway --port=7000 --wavdir=/tmp".

The parameters are in detail:

- help: Display the parameter list with short explanations.
- port: Specify the port the APU Gateway will listen for incoming APU connections. The default value is 6789.
- logfile: Enables logging to a file in addition to standard output.
- loglevel: Sets the level for logging messages from 0 (all log messages) to 5 (no messages). The level names in detail from 0 to 5 are: trace, debug, info, warn, error and fatal.
- wavdir: Specifies a directory for sound data output. Sound data from the APUs (training data, unknown sample data) is saved to this directory in a subdirectory per APU.
- priv: Enable privacy mode. In an insecure environment where sound data should not be stored to local hard drive this option allows invalidating any received sound data before writing to disk. Mainly for testing purposes.
- dbus: Sets the dbus to use. Available options are system and session. It should not be necessary to change this option as the wrapper scripts uses the session DBus as default and if none is available creates one exclusively for the Gateway.
- ssl: Enables or disables (options: on or off) SSL encryption of the network traffic between APU Gateway and APU. APU and Gateway have to use the same setting of course. Default setting is "on".

6.1.4 Graphical user interface

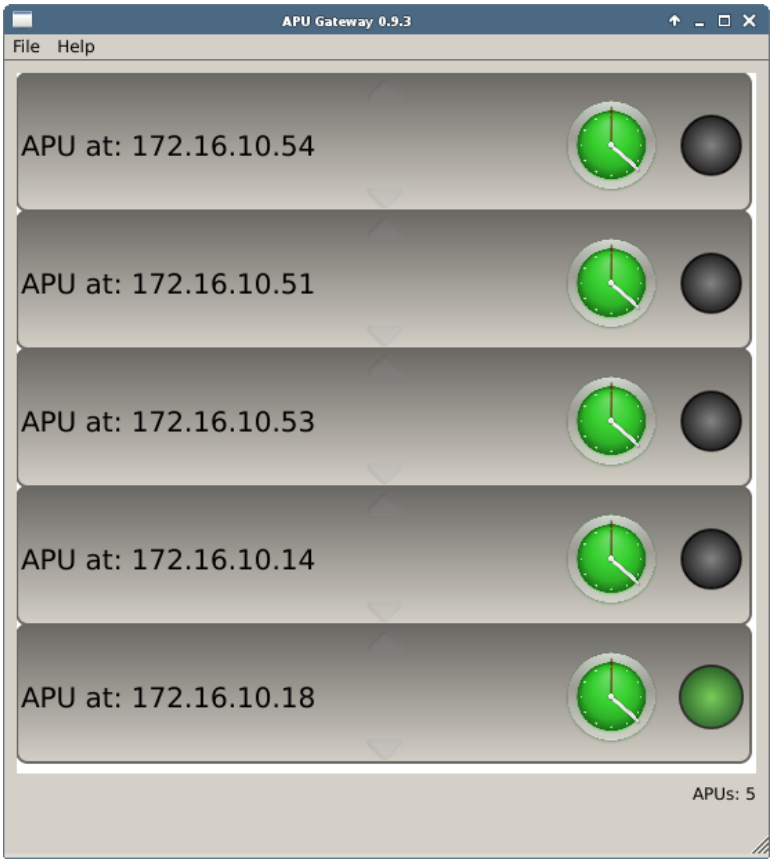

Figure 42: APU gateway graphical user interface

The purpose of the graphical user interface is mainly to give a fast overview about the APUs currently connected to the Gateway and their status. Figure 42 shows the main window of the Gateways user interface with currently 2 connected APUs. Each APU window can be expanded (Figure 43). In this more detailed view live occupancy annotations and the status of to the APU connected ASUs and their microphones is shown (if the respective PlugIns for the AdatReader are enabled, see Section 4.4.2).

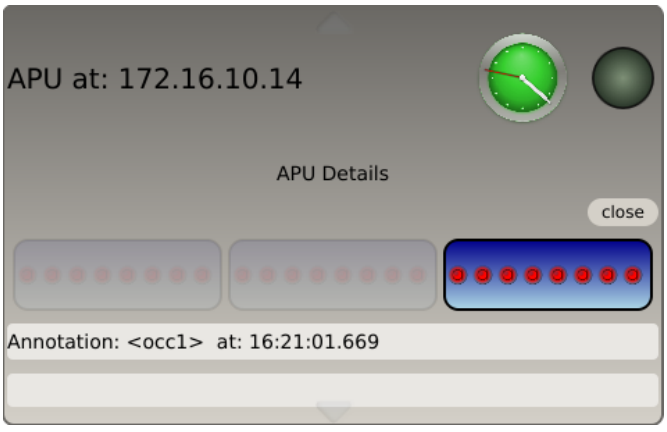

Figure 43: APU gateway GUI expanded view

6.2 PTPd

To synchronize all APU local clocks the BEMO server is required to run the Precision Time Protocol daemon 2 in server mode. Recommended for installation and configuration is the use of the distributions package manager and configuration utilities. If the daemon is not available through the distributions repositories, it can be obtained as source code archive from http://ptpd.sourceforge.net/ (recommended version: 2.2.2). After extracting and compiling the daemon it should be run (only possible as root user) with to following options set:

```
# ./ptpd2 -G -C -b <network interface>
```

6.3 APU management

6.3.1 Network connection

Each APU can be reached for maintenance via an SSH connection on standard port 22. Login as root user with the correct password for the APU can be achieved from any terminal which is able to reach the APU via network with:

```
# ssh root@<APU IP address>
```

6.3.2 Network settings

Network settings of the APU are stored in EEPROM data to ensure persistence throughout firmware updates or booting the rescue system. Therefore the settings can not be changed using standard configuration tools or via editing the interface settings file on the APU file system. Instead two command line utilities are provided for setting and reading network configuration date to/from EEPROM on the APU itself:

netconf2eeprom and eeprom2netconf.

Netconf2eeprom has to be called with several options. Starting the utility without will display this list, too.

```
# netconf2eeprom <dhcp (1|0)> [<address> <netmask> <network>
<gateway> <dns> <bemo>]
```

Only static IP address settings are supported currently, so as an example the following command line would set the APU IP address to 172.16.10.14, with the BEMO server at 172.16.10.203, Gateway and name server at 172.16.10.1:

```
root@apu:~#  netconf2eeprom 0 172.16.10.14 255.255.255.0
172.16.10.0 172.16.10.1 172.16.10.1 172.16.10.203
```

Success of the operation can be validated by reading the network settings from EEPROM using the eeprom2netconf tool:

```
root@apu:~# eeprom2netconf
```

```
auto eth0
iface eth0 inet static
     address 172.16.10.14
     netmask 255.255.255.0
     network 172.16.10.0
     gateway 172.16.10.1
     dns-nameserver 194.95.133.7
     #bemo-server 172.16.10.203
```

After changing network configuration the APU has to be rebooted for the changes to take effect.

6.3.3 Service handling

APU software responsible for the complete APU operation is started using systemd service infrastructure. Notably adatReader, APU-Daemon and FPGA configuration can be managed using systemd's control utility systemctl. Among many other options systemctl allows starting and stopping these services and configures their startup behavior at boot time of the APU.

For example reconfiguring the FPGA can be done with the following command:

```
root@apu:~# systemctl restart fpgaconf.service
```

To enable automatic adatReader startup at boot time this command can be used:

```
root@apu:~# systemctl enable adatReader.service
```

Table 17 shows the APU specific services available with a short description.

Service	Description
adatReader.service	Manages adatReader program startup
apud.service	Manages APU Daemon startup
eeprom2netconf.service	Generates network configuration file from EEPROM data
fpgaconf.service	Upload FPGA configuration
init5_net.service	Configures APU network interface
ptpd.service	Manages PTP daemon startup
watchdog.service	Manages watchdog daemon startup

Table 17: APU services

6.3.4 FPGA configuration update

The FPGA configuration consists of a bitfile located in the APU file system: /opt/s4ecob/res/s4ecob_top.bin. To update the FPGA configuration this file can be simply exchanged. The actual FPGA configuration update is carried out at next APU boot or can be triggered manually be restarting the appropriate service (Section 6.3.3):

```
root@apu:~# systemctl restart fpgaconf.service
```

6.3.5 AdatReader configuration

To start the adatReader process a wrapper script is used: /opt/s4ecob/bin/adatReader.sh. To configure adatReader startup, for example plugin settings, this script can be edited accordingly. Changes take effect after rebooting the APU or restarting the adatReader manually using its systemd service:

```
root@apu:~# systemctl restart adatReader.service
```

6.3.6 Firmware update

Updates of the APU firmware are carried out remotely by booting into the APU's rescue system and exchanging the root file system of the SD card partition containing the normally operational firmware. Several steps are required:

- Login remotely into the APU which should be updated using a secure shell (for example from the BEMO server):

  ```
  bemo # ssh root@<apu-ip-address>
  ```

- The boot partition containing the boot loader configuration has to be mounted:

  ```
  root@apu:~# mount /dev/mmcblk0p1 /mnt
  ```

- Boot settings have to be changed by editing the boot loaders configuration file and setting the variable uenvcmd to "run rescueboot", so the APU will boot into rescue mode. For example by issuing the following command:

  ```
  root@apu:~# sed -i 's/uenvcmd=.*/uenvcmd=run rescueboot/'
  /mnt/uEnv.txt
  ```

- Now the boot partition may be unmounted and the APU rebooted:

  ```
  root@apu:~# umount /mnt; reboot
  ```

- After renewed login to APU and verification that the rescue system was successfully booted (login screen should show "!! Rescue !!" in ASCII art letters), the system partition of the SD card can now be re formated and mounted:

  ```
  root@apu:~# mkfs.ext3 /dev/mmcblk0p3

  root@apu:~# mount /dev/mmcblk0p3 /mnt
  ```

- Now the new firmware image can be extracted to the mounted system partition by for example using a secure shell pipe from the BEMO server:

```
bemo # cat apu_rootfs_<timestamp>.tar.bz2 | ssh root@<apu-ip-address> "tar -x -C /mnt -j -f -"
```

- After unmount of the system partition, the boot loader entry has to be changed back to booting into the normal (now updated) firmware. After that the APU can be rebooted:

```
root@apu:~# umount /mnt

root@apu:~# mount /dev/mmcblk0p1 /mnt

root@apu:~# sed -i 's/uenvcmd=.*/uenvcmd=run secure_boot/' /mnt/uEnv.txt

root@apu:~# umount /mnt; reboot
```

6.3.7 Software update

Updating only parts of the APU software stack is simply done by replacing the individual binary files of the components ready to be updated and restarting the service in question.

7 CONCLUSIONS

The purpose of this document was to describe the design and implementation of the APU hardware and software platform as well as to document the results of corresponding technical tests and measurements. Necessary requirements and specifications were given in various deliverables from WP2 but mainly in Deliverable D2.3. Additional specifications and design details from Deliverables D3.1 and D4.1 had to be considered as well.

After giving an overview over the architecture and general information flow in the overall S4ECoB system solution the architecture of the APU is discussed within this deliverable. Afterwards the APU hardware platform consisting of a mainboard and an FPGA-based extension board is explained in detail. Specifications of all relevant interfaces are listed in the document. FPGA firmware, housing details and hardware cost aspects are addressed as well.

The APU software platform consisting of an embedded operating system, a signal processing framework and a number of communication and synchronization components is introduced afterwards. Software implementation aspects are discussed where necessary.

Furthermore, results of the technical system validation and corresponding tests, which had to be performed to validate the defined requirements from Deliverable D2.3, are reported in this deliverable. A sensor network consisting of up to 7 APUs and connected to an APU gateway was installed in IMMS' labs and afterwards continuously tested over a period of more than three months. In summary, it is documented that the APU fulfills all of the technical requirements necessary for the realization of the S4ECoB demonstrator installations.

Finally, this deliverable contains the APU and occupancy sensor network installation and commissioning manual targeted at installation staff and IT personnel working at the demo site facilities.

REFERENCES

[1] http://www.pandaboard.org/
[2] S.W. Smith, The Scientist and Engineers Guide to Digital Signal Processing
[3] http://www.imms.de/en/competencies/system_design/results/base_box.html
[4] http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/index.htm
[5] http://www.openembedded.org/
[6] Spartan-6 FPGA Configuration Users Guide UG380 (v2.5)
[7] http://www.armadeus.com/wiki/index.php?title=FPGA_loader
[8] http://www.sitime.com/support2/documents/AN10007-Jitter-and-measurement.pdf

ANNEX A: EXPANSION BOARD SCHEMATICS

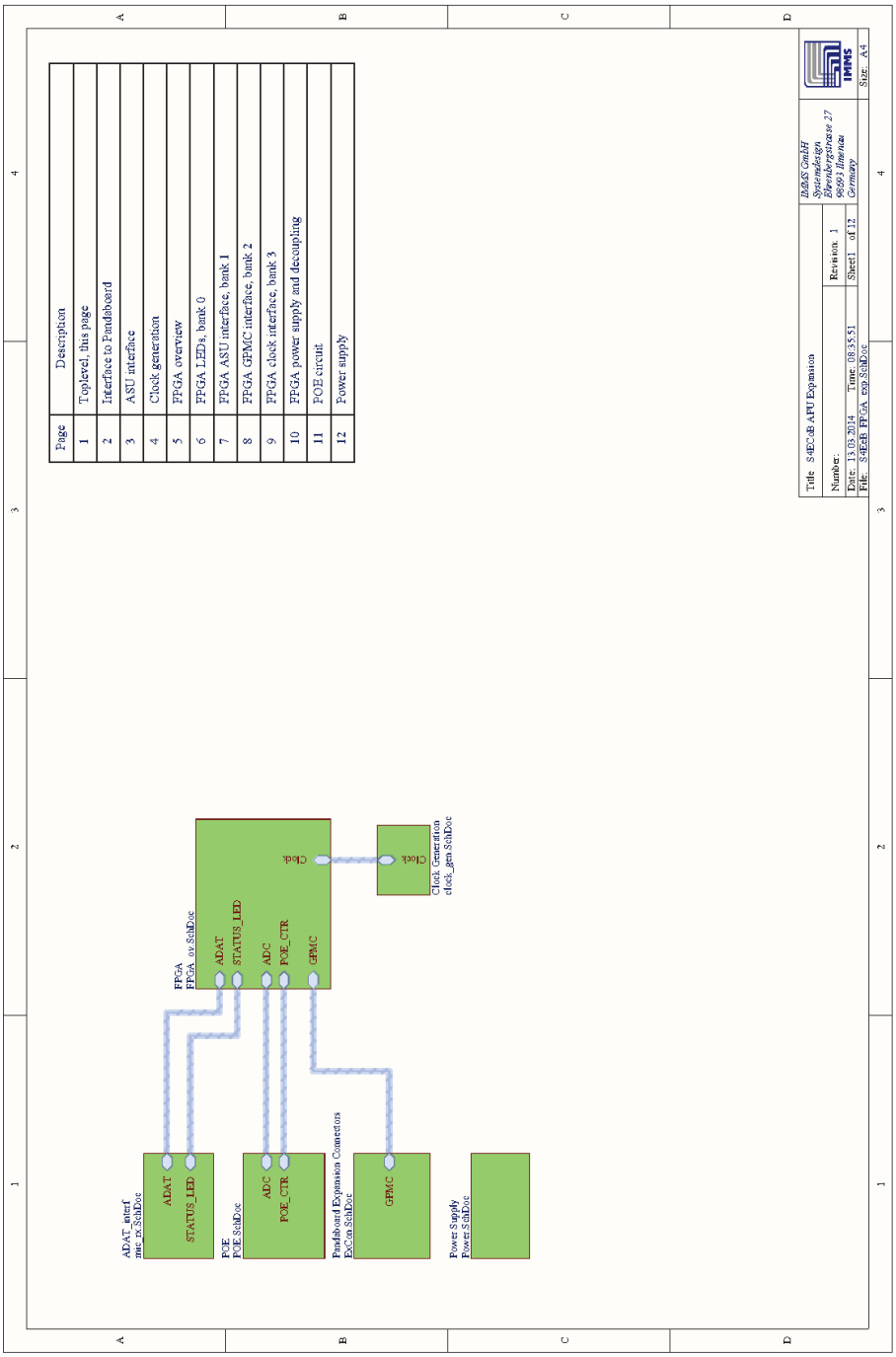

Figure 44: Top-level view

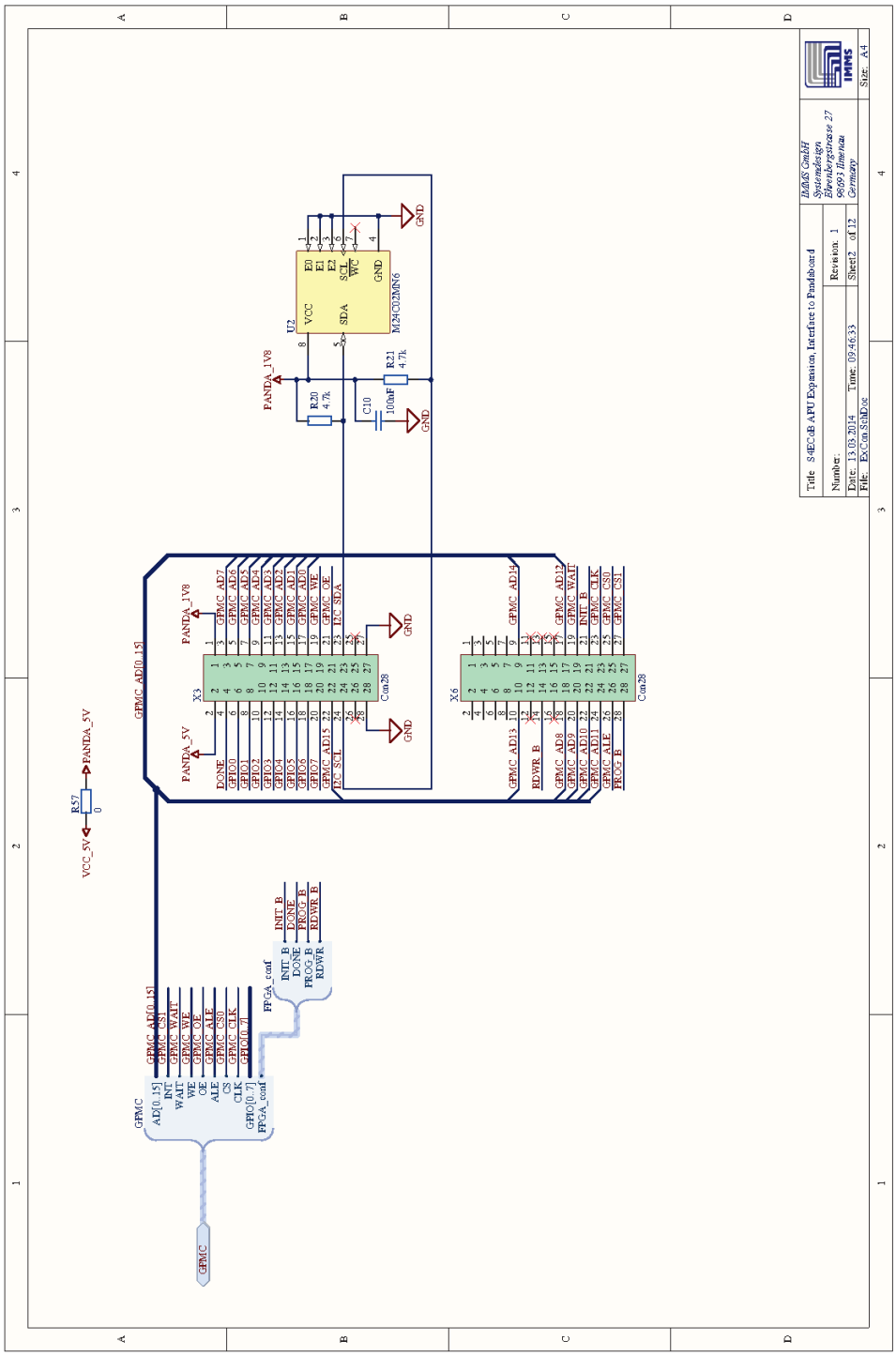

Figure 45: Interface to mainboard

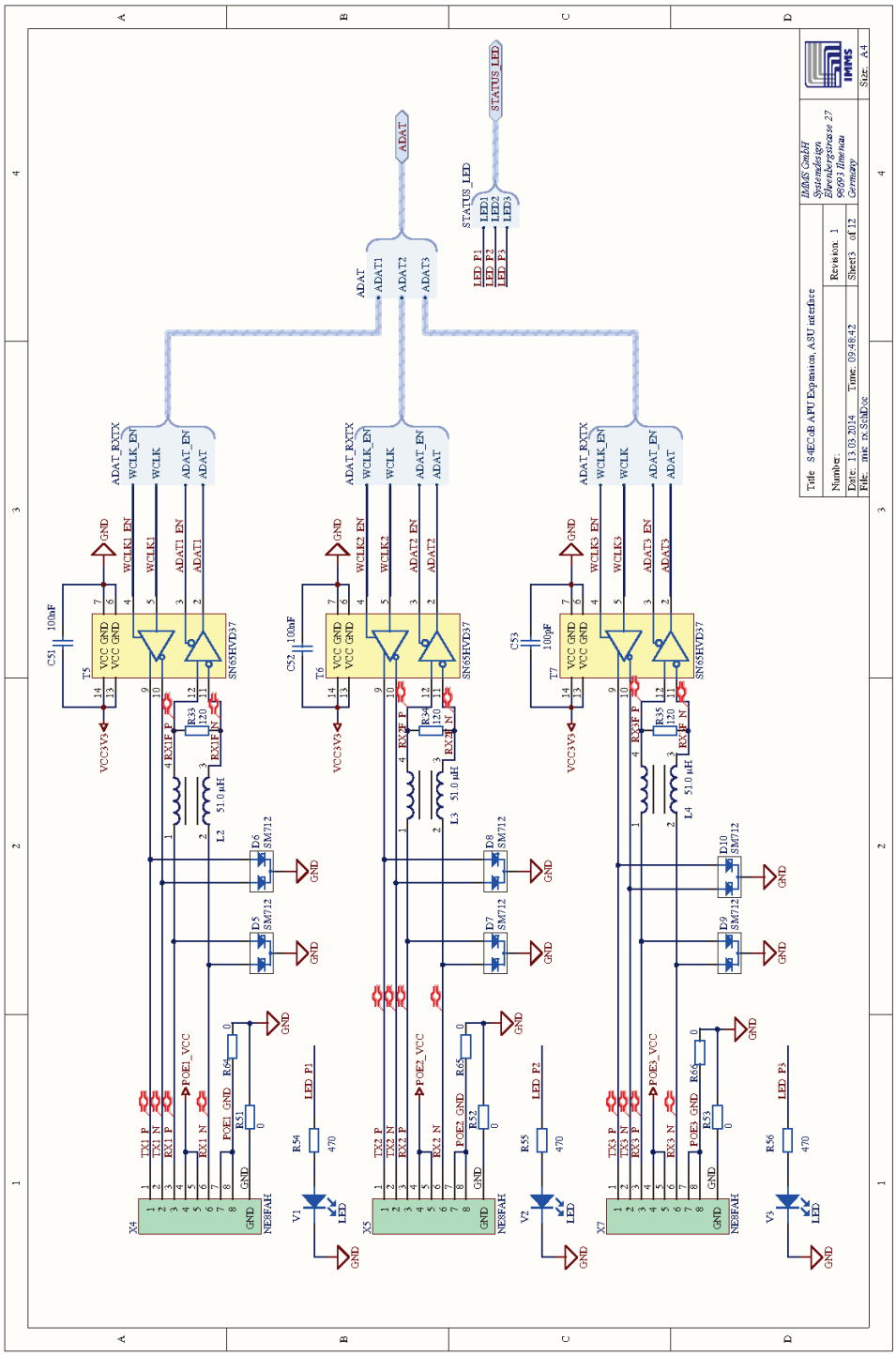

Figure 46: ASU interface

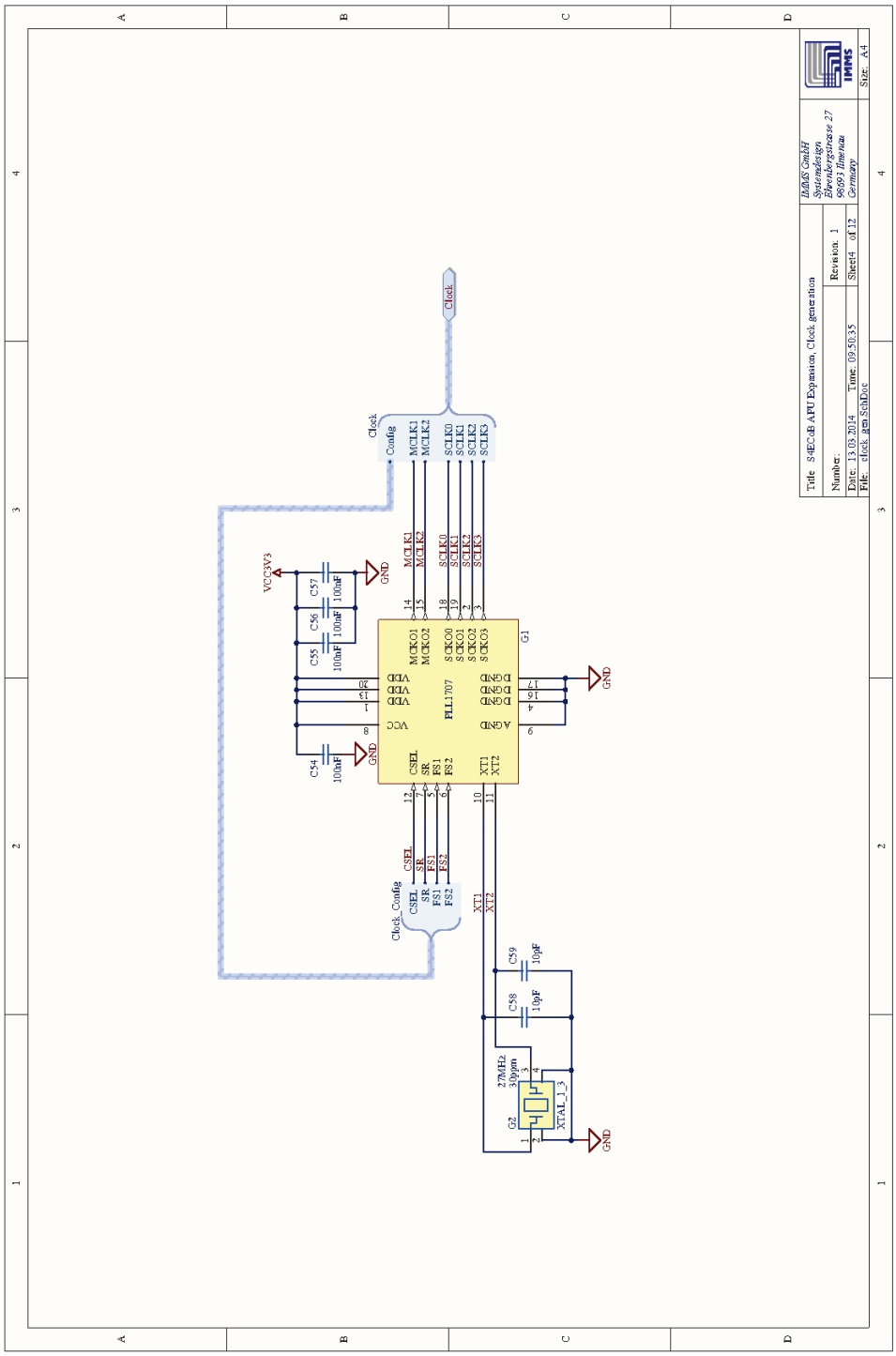

Figure 47: Clock generation

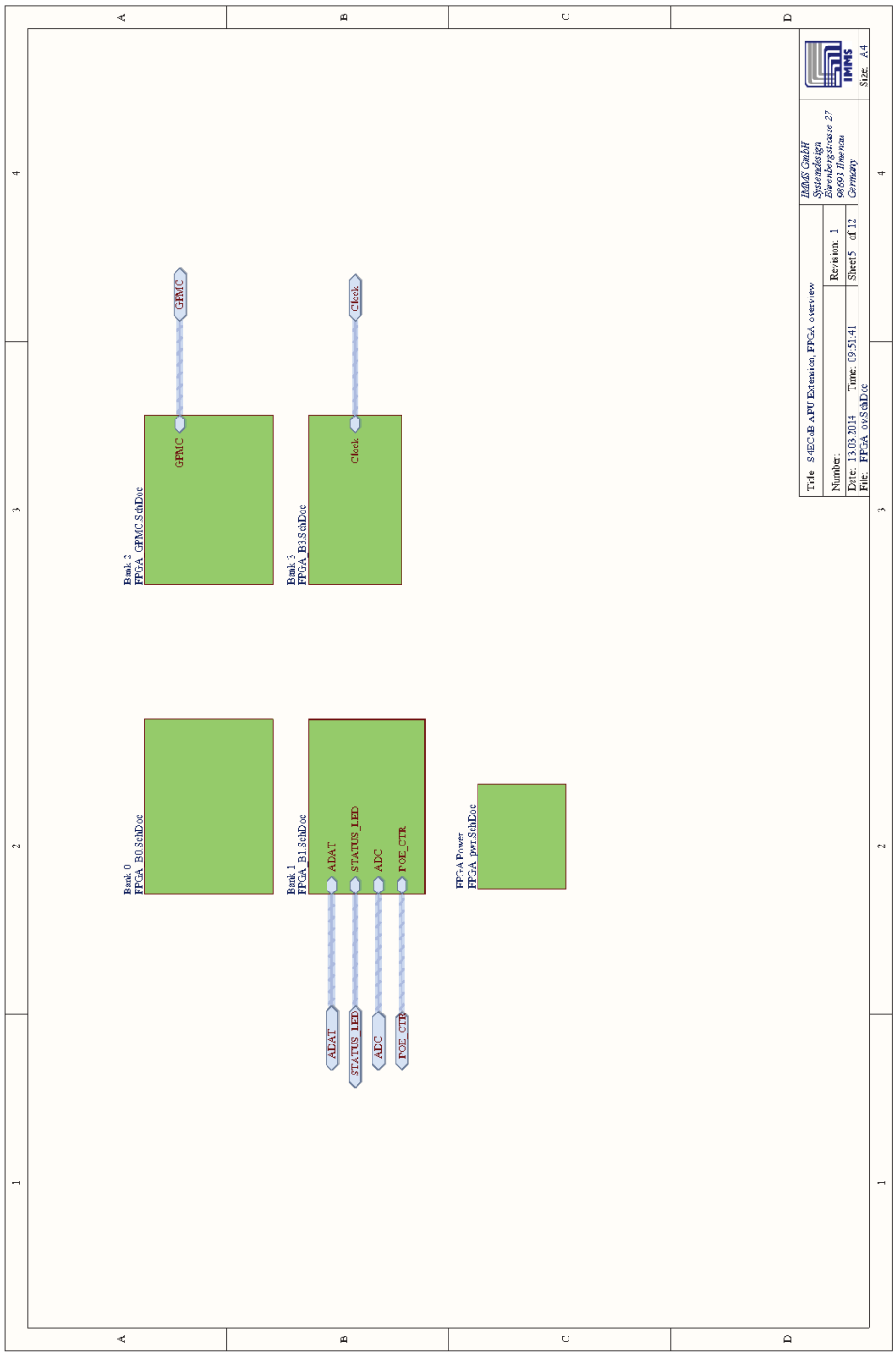

Figure 48: FPGA overview

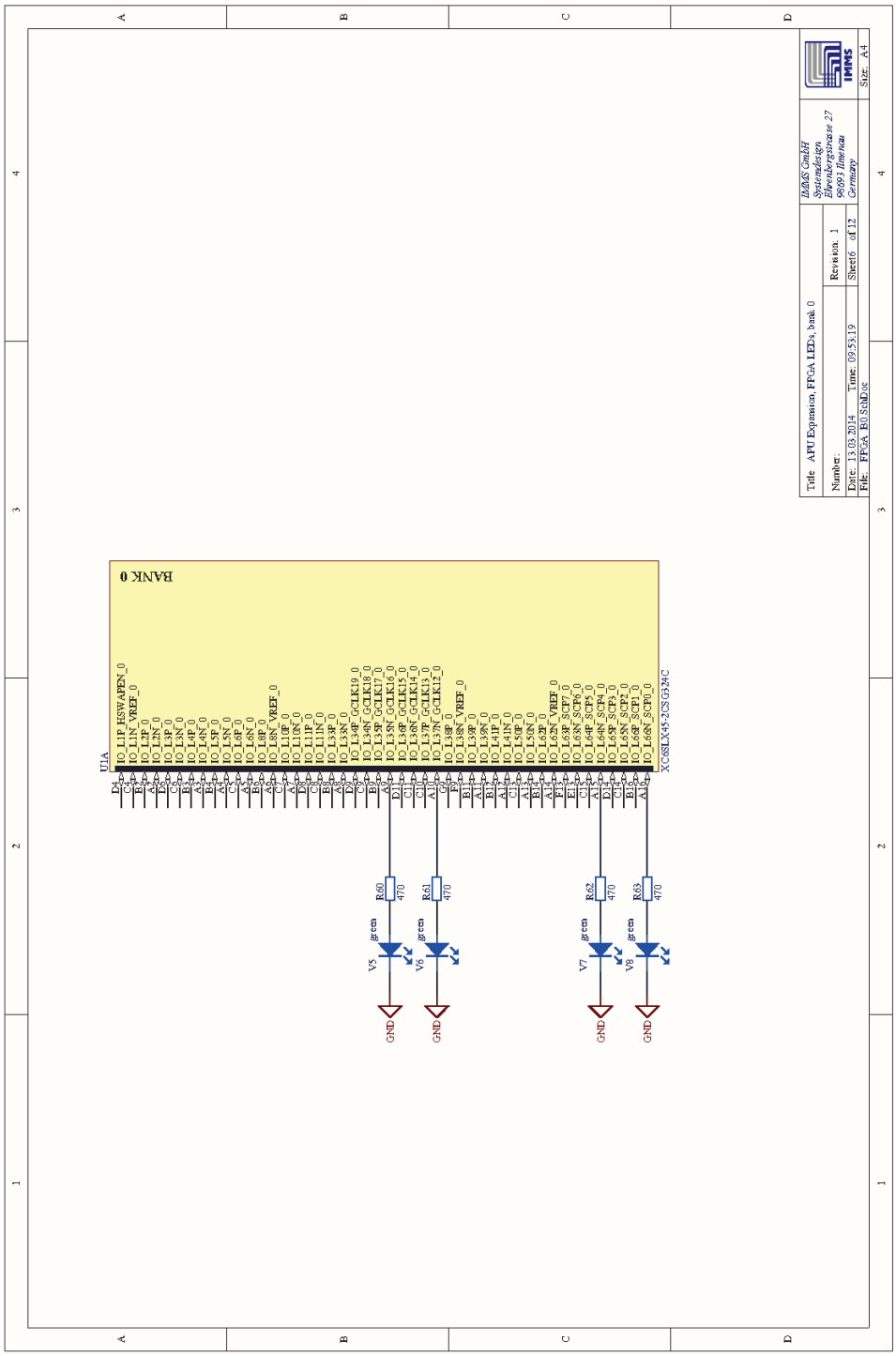

Figure 49: FPGA LEDs

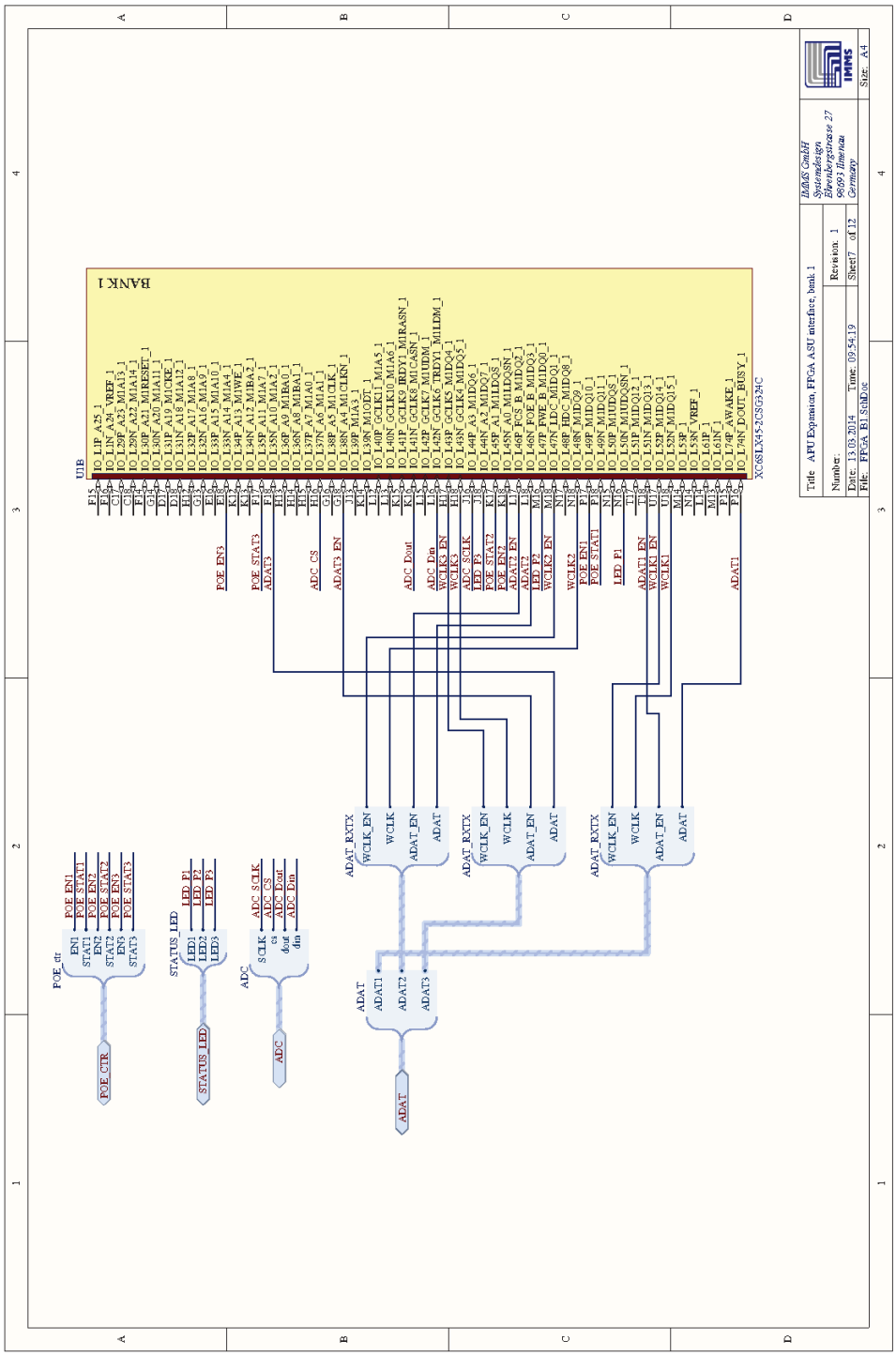

Figure 50: FPGA ASU interface

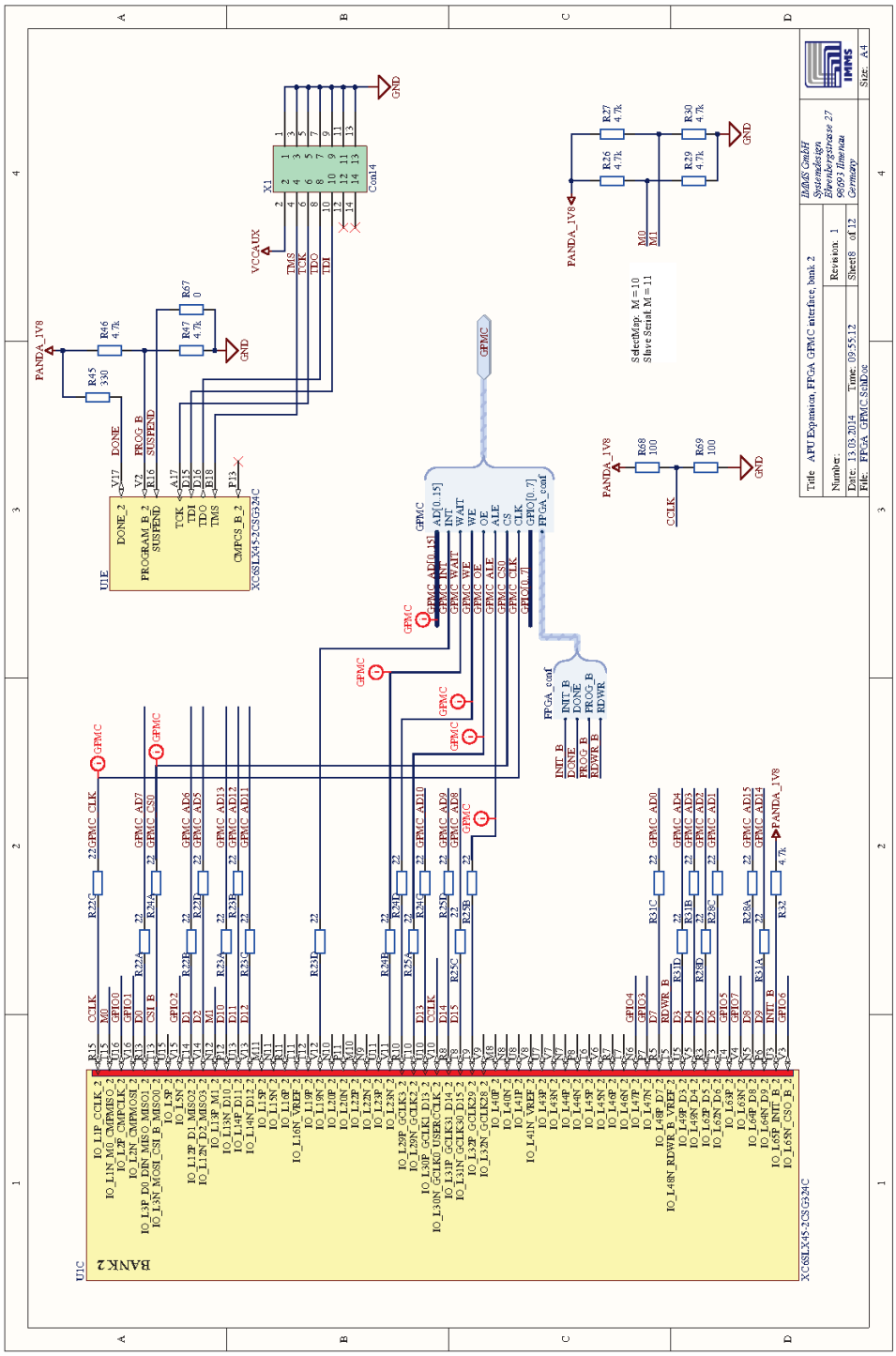

Figure 51: FPGA GPMC interface

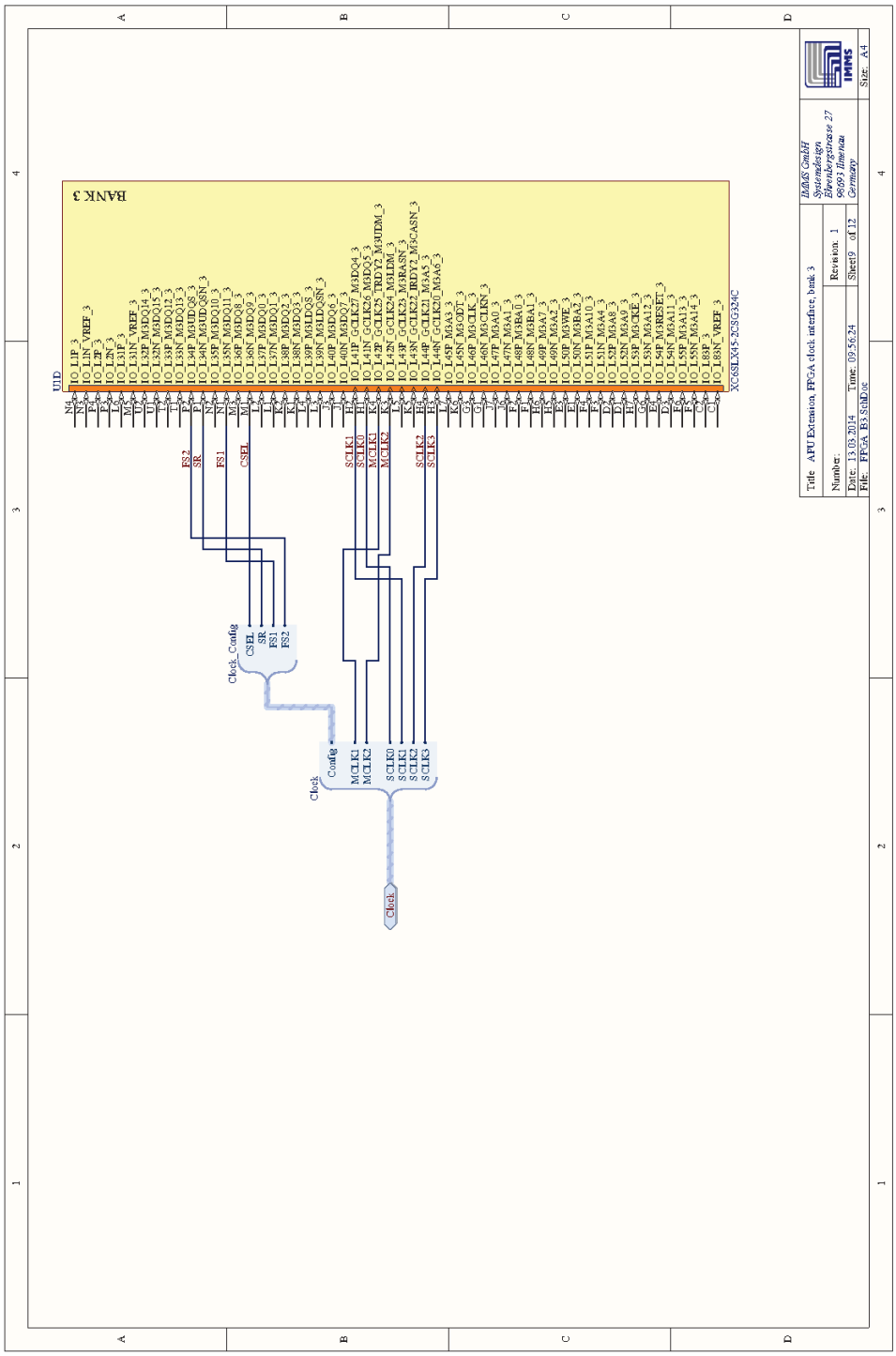

Figure 52: FPGA clock interface

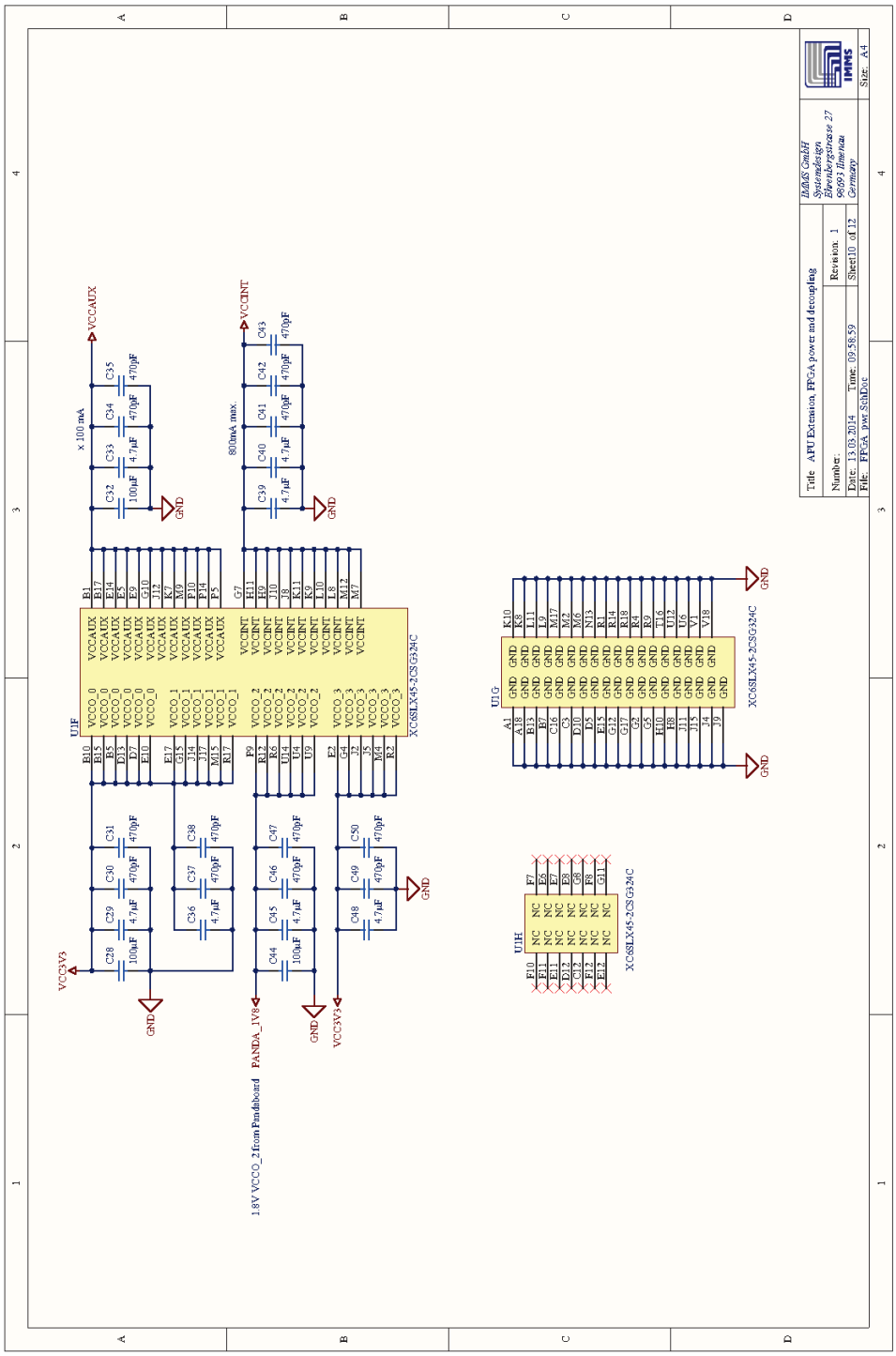

Figure 53: FPGA power supply and decoupling

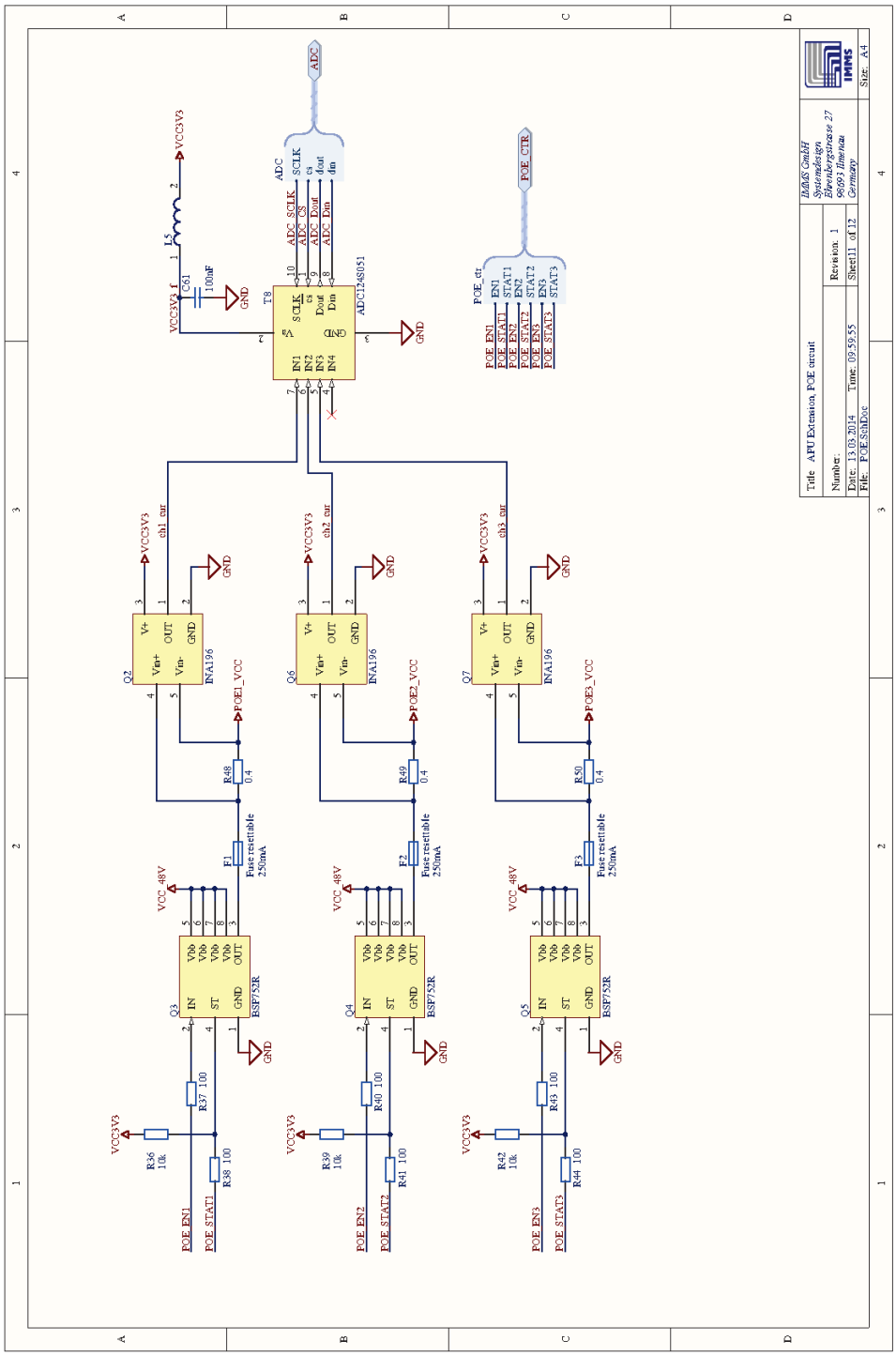

Figure 54: PoE circuit

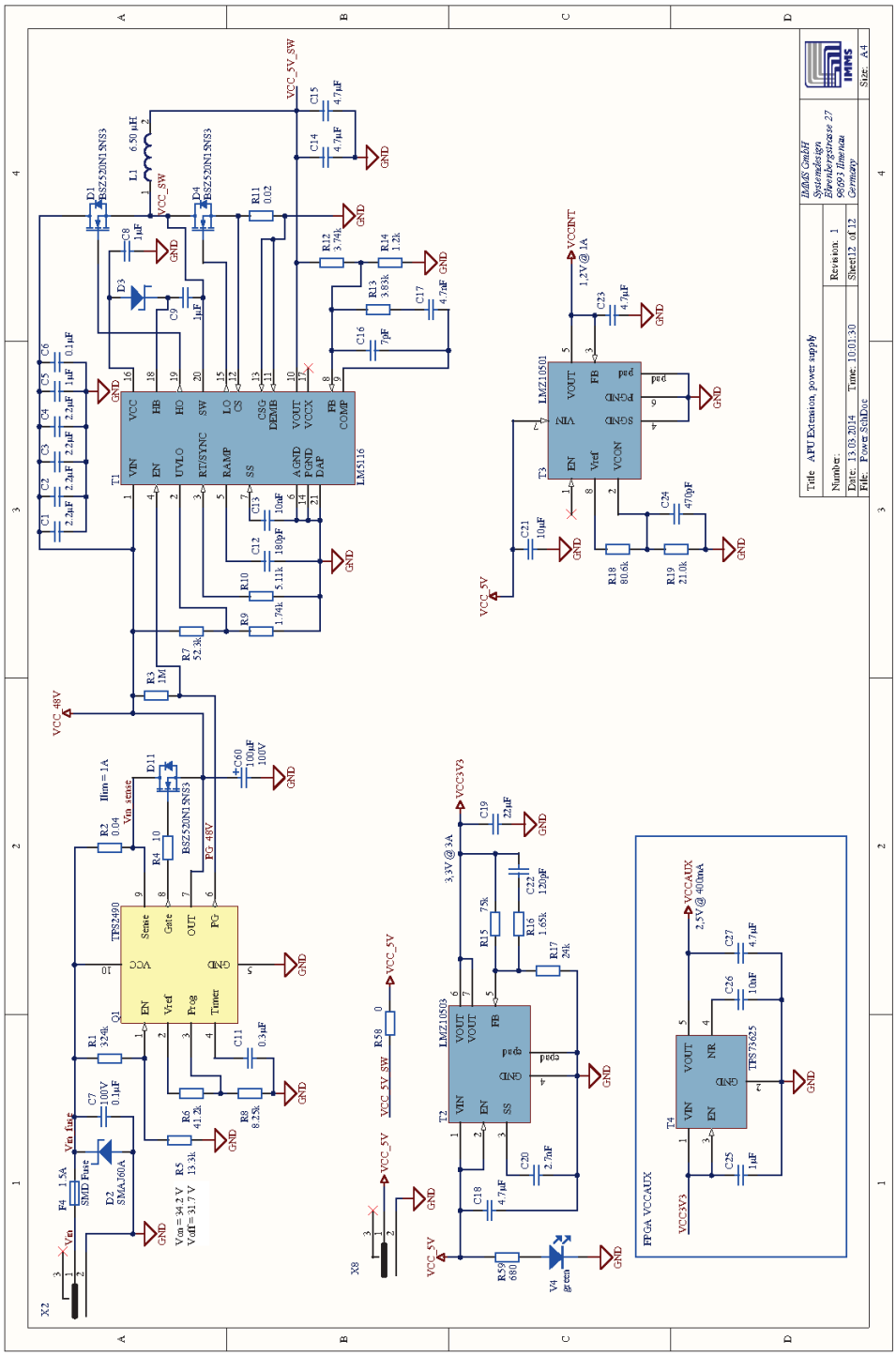

Figure 55: Power supply

ANNEX B: APU PHYSICAL AND ELECTRICAL INSTALLATION MANUAL

Steps of the installation and commissioning process

- Mounting
- Electrical connections
- APU gateway installation (this has to be done before the functional test - see Section 6)
- Functional test

Mounting

There are two possible options for the APU fixing. The APU can directly screwed using the holes in the housing. The second option is to use the delivered wall fastening clip, so the APU can be fixed without removing the APU housing cover.

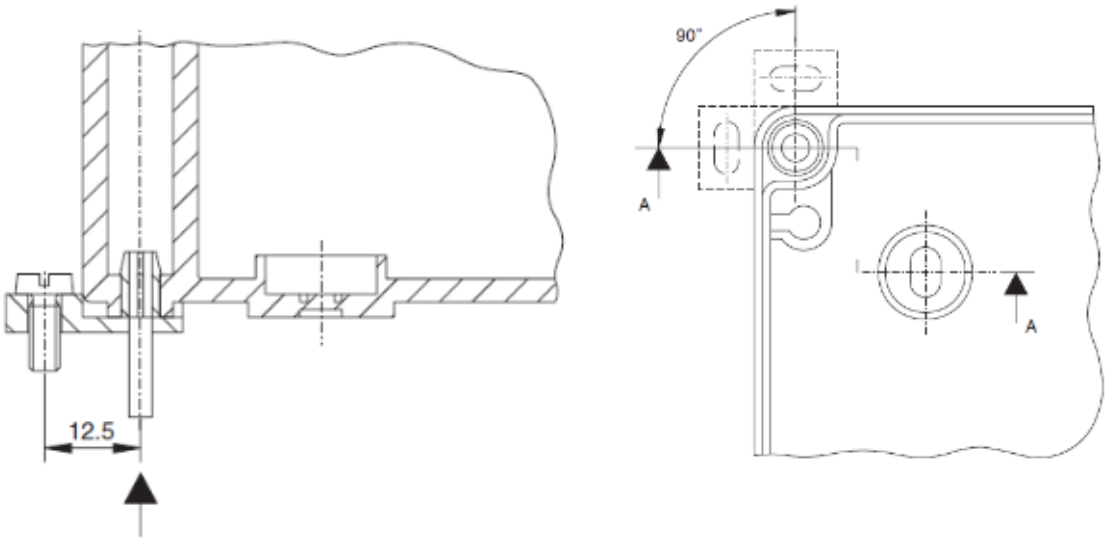

Figure 56: APU mounting

Interfaces

- ASU interface

Connect the Cat 5 ASU cables to the ASU interface connectors as shown in Figure 57.

ATTENTION: Do not connect a LAN network Ethernet cable to one of these ports!

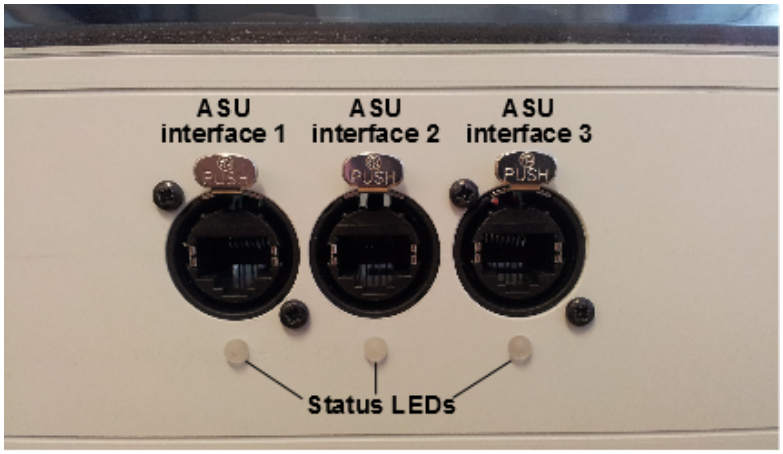

Figure 57: ASU interface connectors of the APU

- LAN interface

Connect the Ethernet cable with APU LAN interface.

- Power supply

Connect the APU with the delivered 5V wall plug power supply.

ATTENTION: Connecting a supply with an output higher than +5Vdc could cause possible damage!

Figure 58: LAN interface and power supply connector of the APU

Functional test

The green power LED is turned on once the APU is supplied with power. If ASU and APU are connected and both are turned on then the status LED of each ASU interface should show successful data transfer.